(2)

ANNUAL REPORT

VOLUME 3

PART 2

TASK 3: SPECIAL STUDIES

REPORT NO. AR-0142-91-002

September 27, 1991

GUIDANCE, NAVIGATION AND CONTROL

DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332-0540

Contract Data Requirements List Item A005

Period Covered: FY 91

Type Report: Annual

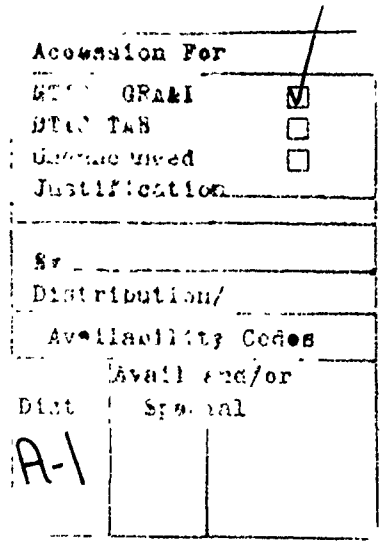
DISCLAIMER

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) This material is approved for public release unlimited distribution.

(2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227–7013, October 1988.



ANNUAL REPORT

VOLUME 3

PART 2

TASK 3: SPECIAL STUDIES

September 27, 1991

Authors

Cecil O. Alford, Richard M. Pitts and Philip R. Bingham

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332-0540

Eugene L. Sanders	**Cecil O. Alford**
USASDC	**Georgia Tech**
Contract Monitor	**Project Director**

TABLE OF CONTENTS

List of Figures

VOLUME 3

PART 2

TASK 3: SPECIAL STUDIES

6. Appendix A: EXOSIM v1.0 Boost Phase

A.1 Mainline (FORTRAN)

A.1.1 Aeroca.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL ca1m(205), ca2m(205)
      REAL mach, alfat, ca
#include "include/constant.dat"
#include "include/ca1m.dat"
#include "include/ca2m.dat"
      DATA icam1/0/, icaa1/0/
      DATA icam2/0/, icaa2/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_net_32(mach)
      CALL receive_net_32(alfat)

      IF (t .GT. tstg1) THEN
c second stage
        CALL tlu2ei(mach, alfat, ca2m, icam2, icaa2, ca)
      ELSE
c first stage
        CALL tlu2ei(mach, alfat, ca1m, icam1, icaa1, ca)
      ENDIF

      CALL send_net_32(ca)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.2 Aerocn.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL cna1(205), cna2(205)
      REAL mach, alfat, cn
#include "include/constant.dat"
#include "include/cna1.dat"
#include "include/cna2.dat"
      DATA icnm1/0/, icna1/0/
      DATA icnm2/0/, icna2/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(mach)
      CALL receive_real_32bit(alfat)

      IF (t .GT. tstg1) THEN
c second stage
        CALL tlu2ei(mach, alfat, cna2, icnm2, icna2, cn)
      ELSE
c first stage
        CALL tlu2ei(mach, alfat, cna1, icnm1, icna1, cn)
      ENDIF

      CALL send_real_32bit(cn)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.3 Aeroxcp.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL xcpl1(205), xcpl2(205)
      REAL mach, alfat, xcp
#include "include/constant.dat"
#include "include/xcpl1.dat"
#include "include/xcpl2.dat"
      DATA ixcpm1/0/, ixcpa1/0/
      DATA ixcpm2/0/, ixcpa2/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(mach)
      CALL receive_real_32bit(alfat)

      IF (t .GT. tstg1) THEN
c second stage
        CALL tlu2ei(mach, alfat, xcpl2, ixcpm2, ixcpa2, xcp)
      ELSE
c first stage
        CALL tlu2ei(mach, alfat, xcpl1, ixcpm1, ixcpa1, xcp)
      ENDIF
      xcp =  - xcp/12.0

      CALL send_real_32bit(xcp)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.4 Attlm.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL attltt(5), attlmt(5)
      REAL attlm
#include "include/constant.dat"
#include "include/attltt.dat"
#include "include/attlmt.dat"
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL table(attltt, attlmt, t, attlm, 5, itable)

      CALL send_real_32bit(attlm)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.5 Bauto.for

```
      PROGRAM main
c-----------------------------------------------------------------------
c      subroutine bauto(t,thter,psier,sq,sr,ii,cgest,vrwm,alt,cmmd,
c     .                 dlpc,dlyc,mdltfr,malpha)
c-----------------------------------------------------------------------
c     function :            provides control of the missile about three
c                           axes throughout the boost phase of flight
c     inputs :              t,thter,psier,sq,sr,ii,cgest,vrwm,alt
c     outputs :             cmmd,dlpc,dlyc,mdltfr,malpha
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL cgest(3), ii(3), alt, cne, xcpe
      REAL estmch, alfate
      REAL cmmd(2)
      REAL ktht, kpsi, kthtd
      REAL kpsid, timte1(26), timte2(29)
      REAL thrte1(26), thrte2(29), altte(59)
      REAL rhote(59), vrwm(3)
      REAL ld, kne, kme
      REAL lfracs, frcloc(3, 4)
      REAL kthtk1, kthtk2
      REAL krtfrc, kthfrc
      REAL kthfm1, kthfm2, vsndte(59)
      REAL presste(59), mchlim, wmfrtt(4)
      REAL wmfrct(4)
      REAL dlpc, dlyc, sq, sr, mdltfr, malpha
#include "include/constant.dat"
#include "include/frcloc.dat"
#include "include/altte.dat"
#include "include/rhote.dat"
#include "include/presste.dat"
#include "include/vsndte.dat"
#include "include/timte1.dat"
#include "include/timte2.dat"
#include "include/thrte1.dat"
#include "include/thrte2.dat"
      DATA tapu/0.0/, dtapu/0.005/, tapustep/5.0/
      DATA mchlim/4.0/, kthtk1, kthtk2/.6, 1.5/
      DATA tign/0.01/, tst2on/22.995/, tfrcs/23.0/, tmode2/23.01/
      DATA wmtvc/25.0/, zettvc/0.85/, wmfrtt/0.0, 9.5, 39.95, 100./
      DATA wmfrct/62.83, 62.83, 42., 42./, zetfrc/0.85/, delon/0.045/
      DATA bcklmt/0.15/, delthg/0.045/, thjet/370./, sjet/1.3273/
      DATA sref1/1.968953/, sref2/1.968953/, aexite/.305/, aexit2/0.99/
      DATA xnoze/-12.5583/, xnoz2/-7.39167/, djet/1.3/, xjet/-2.71/
      DATA ialte/0/, ith1e/0/, ith2e/0/, iwmfrc/0/
      DATA dtr/0.017453292519943296/, slglbm/32.174048/

c initialize time
      tstep = 0.0
      t = tstep * delt

      cmmd(1) = 0.0
      cmmd(2) = 0.0
      dlpc = 0.0
      dlyc = 0.0
      mdltfr = 0.0
      malpha = 0.0
      sq = 0.0
      sr = 0.0
```

```
 10 CONTINUE
    CALL send_real_32bit(cmmd(1))
    CALL send_real_32bit(cmmd(2))
    CALL send_real_32bit(dlpc)
    CALL send_real_32bit(dlyc)
    CALL send_real_32bit(sq)
    CALL send_real_32bit(sr)
    CALL send_real_32bit(mdltfr)
    CALL send_real_32bit(malpha)
    CALL receive_real_32bit(cgest(1))
    CALL receive_real_32bit(cgest(2))
    CALL receive_real_32bit(cgest(3))
    CALL receive_real_32bit(ii(2))
    CALL receive_real_32bit(alt)
    CALL receive_real_32bit(vrwm(1))
    CALL receive_real_32bit(vrwm(2))
    CALL receive_real_32bit(vrwm(3))

    IF (tstep .GE. tapu) THEN
      tapu = tapu + tapustep

      IF (t .LT. tstg2) THEN
        IF (abs(t - tstg1) .LE. dteps) THEN
          aexite = aexit2
          xnoze = xnoz2
        ENDIF
        estalt = alt
        CALL table(altte, rhote, estalt, estrho, 59, ialte)
        estrho = estrho*1.0e-6/slglbm
        CALL table(altte, presste, estalt, estpre, 59, ialte)
        CALL table(altte, vsndte, estalt, estvsd, 59, ialte)
        estvel = sqrt(vrwm(1)**2 + vrwm(2)**2 + vrwm(3)**2)
        estmch = estvel/estvsd
        estqa = estrho*estvel**2/2.0
        IF (t .GT. tstg1) THEN
          srefe = sref2
          t0 = t - tst2on
          CALL table(timte2, thrte2, t0, thrve, 29, ith2e)
        ELSE
          t0 = t - tign
          srefe = sref1
          CALL table(timte1, thrte1, t0, thrve, 26, ith1e)
        ENDIF
        thre = thrve - aexite*estpre
        IF (thre .LT. 0.0) thre = 0.0
        IF (estvel .GT. 0.0) THEN
          alfate = sparctan(sqrt(vrwm(2)**2 + vrwm(3)**2), abs(vrwm
     &      (1)))/dtr
        ELSE
          alfate = 0.0
        ENDIF

        CALL send_real_32bit(estmch)
        CALL send_real_32bit(alfate)
*         if ( t.lt.tstg1 ) then
*            call tlu2ei(estmch,4.0d0,cna1e,icnm1e,icna1e,cne)
*            call tlu2ei(estmch,alfate,xcpl1e,icpm1e,icpa1e,xcpe)
*         else
*            call tlu2ei(estmch,4.0d0,cna2e,icnm2e,icna2e,cne)
*            call tlu2ei(estmch,alfate,xcpl2e,icpm2e,icpa2e,xcpe)
*         end if
        CALL receive_real_32bit(cne)
        CALL receive_real_32bit(xcpe)
```

```
c  conversion from inches to feet
          xcpe =  - xcpe/12.0
c  calculate cnalfa (per radian)
          cnalp = cne/(4.0*dtr)
          xcpcg = xcpe - cgest(1)
          IF (thre .GE. 1000.0 .AND. ii(2) .GT. 1.0e-6) malpha = abs
     &     (cnalp*xcpcg*srefe*estqa/ii(2))

          CALL receive_real_32bit(psier)
          CALL receive_real_32bit(thter)
          CALL receive_real_32bit(sq)
          CALL receive_real_32bit(sr)

c  tvc autopilot
          IF (t .LT. tmode2) THEN
            IF (thre .GE. 1000.0 .AND. ii(2) .GT. 1.0e-6) THEN
              xdel = cgest(1) - xnoze
              ktht = (ii(2)*wmtvc**2 + cnalp*srefe*estqa*xcpcg)/(thre
     &         *xdel)
              kpsi = (ii(2)*wmtvc**2 + cnalp*srefe*estqa*xcpcg)/(thre
     &         *xdel)
              kthtd = 2.0*zettvc*wmtvc*ii(2)/(thre*xdel)
              kpsid = 2.0*zettvc*wmtvc*ii(2)/(thre*xdel)
            ELSE
              ktht = 4.0
              kpsi = 4.0
              kthtd = 4.0
              kpsid = 4.0
            ENDIF
            cmmd(1) = thter*ktht - sq*kthtd
            cmmd(2) = psier*kpsi - sr*kpsid
            totcmd = sqrt(cmmd(1)**2 + cmmd(2)**2)
            IF (totcmd .GT. bcklmt) THEN
              cmmd(1) = cmmd(1)*bcklmt/totcmd
              cmmd(2) = cmmd(2)*bcklmt/totcmd
            ENDIF
          ELSE
            cmmd(1) = 0.0
            cmmd(2) = 0.0
          ENDIF
c  forward reaction control system autopilot
          IF (t .GE. tfrcs) THEN
            IF (thre .GE. 1000.0 .AND. ii(2) .GT. 1.0e-6) THEN
              ld = (xjet - xnoze)/djet
              ct = thjet/(estqa*sjet)
              IF (estmch .LE. mchlim) THEN
                kne = 0.6118 + (0.1358*(1. - 0.485*sqrt(ld))/sqrt(ct))
     &           + 0.0946*estmch + 0.004317/ld
              ELSE
                kne = 1.0 + exp(1.1 - 0.2116*(log(ct) + 8.5)**1.4)
              ENDIF
              kme = 0.5582 - 0.1884/sqrt(ct) - 1.9659/ld
              lfracs = frcloc(1, 1) - cgest(1)
              mdltfr = ( - kme*thjet*djet + kne*thjet*lfracs)/ii(2)
              CALL table(wmfrtt, wmfrct, t - tmode2, wmfrc, 4,
     &         iwmfrc)
              krtfrc = 2.0*zetfrc*wmfrc/(wmfrc**2 + malpha)
              kthfrc = 2.0*delon/(mdltfr*krtfrc*dtapu)
              kthfm1 = delon*malpha/mdltfr
              kthfm2 = delon/delthg
              IF (kthfrc .LT. kthfm1) kthfrc = kthfm1
              IF (kthfrc .LT. kthfm2) kthfrc = kthfm2
              ktht = kthfrc*kthtk1
              kthtd = ktht*krtfrc*kthtk2
```

```
              kpsi = ktht
              kpsid = kthtd
            ELSE
              malpha = 544.18
              mdltfr = 6.0437
              ktht = 10.0
              kthtd = 25.0
              kpsi = 10.0
              kpsid = 25.0
            ENDIF
            dlpc = thter*ktht - sq*kthtd
            dlyc = psier*kpsi - sr*kpsid
          ENDIF
        ELSE
          CALL send_real_32bit(estmch)
          CALL send_real_32bit(alfate)
          CALL receive_real_32bit(cne)
          CALL receive_real_32bit(xcpe)
          CALL receive_real_32bit(psier)
          CALL receive_real_32bit(thter)
          CALL receive_real_32bit(sq)
          CALL receive_real_32bit(sr)
        ENDIF
      ELSE
        CALL send_real_32bit(estmch)
        CALL send_real_32bit(alfate)
        CALL receive_real_32bit(cne)
        CALL receive_real_32bit(xcpe)
        CALL receive_real_32bit(psier)
        CALL receive_real_32bit(thter)
        CALL receive_real_32bit(sq)
        CALL receive_real_32bit(sr)
      ENDIF

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.6 Boost2a.for

```
      PROGRAM main
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      REAL gr(3)
      DOUBLE PRECISION latlp, longlp
      DOUBLE PRECISION xyz(3)
      DOUBLE PRECISION xyzd(3), xyzdd(3), xyze(3)
      DOUBLE PRECISION xyzed(3), xyzedd(3), cei(9)
      REAL mdotf, msstg2, mass, massold, mass0
      REAL frcx, frcy, frcz, alt
      REAL mdot, fxt, fyt, fzt
      REAL ud, vd, wd, wkv, wkv0, wdotkv
      REAL cim(9), mdott, spt
        REAL tmp_xyz(3)
#include "include/constant.dat"
      DATA rade/20898908.0/, msstg2/19.457/
      DATA dtr/0.017453292519943296/, slglbm/32.174048/
      DATA mass0/43.939/, wkv0/97.1/
      DATA latlp/0.0/, longlp/0.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

      DO 10 i = 1, 3
        xyze(i) = 0.0
        xyzed(i) = 0.0
        xyzedd(i) = 0.0
   10 CONTINUE
      xyze(1) = rade
      alt = dsqrt(xyze(1)**2 + xyze(2)**2 + xyze(3)**2) - rade
c-------------------------------------------------------------------------c
c----------------------------- missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      mass = mass0
      wkv = wkv0
      CALL mmk(-90.0*dtr, 1, latlp*dtr, 2, longlp*dtr, 3, cei)
      CALL vecrot(xyzed, cei, xyzd)
      CALL vecrot(xyze, cei, xyz)
      mdot = 0.0
      wdotkv = 0.0
      CALL vecrot(xyzedd, cei, xyzdd)
      spt = t
      CALL spintegi(mass, mdot, spt, 1)
      CALL spintegi(wkv, wdotkv, spt, 5)
      CALL integi(xyzd(1), xyzdd(1), t, 6)
      CALL integi(xyzd(2), xyzdd(2), t, 7)
      CALL integi(xyzd(3), xyzdd(3), t, 8)
      CALL integi(xyz(1), xyzd(1), t, 9)
      CALL integi(xyz(2), xyzd(2), t, 10)
      CALL integi(xyz(3), xyzd(3), t, 11)
c-------------------------------------------------------------------------
c  initialize processor inputs if not already initialized
c p1
      fxt = 0.0
      fyt = 0.0
      fzt = 0.0
      mdott = 0.0
      frcx = 0.0
      frcy = 0.0
      frcz = 0.0
```

```
      mdotf = 0.0
c p2
c  p3
      ud = 0.0
      vd = 0.0
      wd = 0.0
c  p4
      DO 20 i = 1, 3
        gr(i) = 0.0
   20 CONTINUE
c  p5
c  initialization routine
      mass = mass + delt*mdot
c-------------------------------------------------------------------------c
c---------------------------- main execution loop ------------------------c
c-------------------------------------------------------------------------c
   30 CONTINUE
c*************************************************************************
c                                                          *
c                            partition 1                        *
c                                                          *
c*************************************************************************
c--------------------------- missile state update module ----------------c
c  temporarily extrapolate missile states from last integration
c  step ( note : the extrapolated states are overwritten when
c                the true integration is performed )
c------------- send parameters to partitions 3, 4, and 5 --------------c
      CALL send_real_32bit(ud)
      CALL send_real_32bit(vd)
      CALL send_real_32bit(wd)
      CALL send_real_32bit(gr(1))
      CALL send_real_32bit(gr(2))
      CALL send_real_32bit(gr(3))
      CALL send_real_32bit(alt)
c---------- send mass to masspr subroutine table lookup processors ----c
      CALL send_real_32bit(mass)
      xyzd(1) = xyzd(1) + delt*xyzdd(1)
      xyzd(2) = xyzd(2) + delt*xyzdd(2)
      xyzd(3) = xyzd(3) + delt*xyzdd(3)
      xyz(1) = xyz(1) + delt*xyzd(1)
      xyz(2) = xyz(2) + delt*xyzd(2)
      xyz(3) = xyz(3) + delt*xyzd(3)
c  calculate current missile altitude
      alt = dsqrt(xyz(1)**2 + xyz(2)**2 + xyz(3)**2) - rade
c----------- send altitude to atmos subroutine table lookup processors -c
      CALL send_real_32bit(alt)
c----------- send parameters to thread containing winds subroutine -----c
      CALL send_real_64bit(xyz(1))
      CALL send_real_64bit(xyz(2))
      CALL send_real_64bit(xyz(3))
      CALL send_real_64bit(xyzd(1))
      CALL send_real_64bit(xyzd(2))
      CALL send_real_64bit(xyzd(3))
      CALL receive_real_32bit(cim(1))
      CALL receive_real_32bit(cim(2))
      CALL receive_real_32bit(cim(3))
      CALL receive_real_32bit(cim(4))
      CALL receive_real_32bit(cim(5))
      CALL receive_real_32bit(cim(6))
      CALL receive_real_32bit(cim(7))
      CALL receive_real_32bit(cim(8))
      CALL receive_real_32bit(cim(9))
c---------- receive parameters from partition #2 ---------------------c
      CALL receive_real_32bit(fxt)
```

```
      CALL receive_real_32bit(fyt)
      CALL receive_real_32bit(fzt)
      CALL receive_real_32bit(frcx)
      CALL receive_real_32bit(frcy)
      CALL receive_real_32bit(frcz)
      CALL receive_real_32bit(mdott)
      CALL receive_real_32bit(mdotf)
      wkv = wkv + delt*wdotkv
      mdotkv =  - mdotf*slglbm
      mdot =  - mdott - mdotf
c  save mass value for use in missil subroutine
      massold = mass
      spt = t
c  trapezoidal integration for simplicity
      IF (dabs(t - tstg1) .LE. dteps) THEN
c  first stage separation
        mass = msstg2
        CALL spintegi(mass, 0.0e0, spt, 1)
      ELSEIF (dabs(t - tstg2) .LE. dteps) THEN
c  second stage separation
        mass = wkv/slglbm
        CALL spintegi(mass, 0.0e0, spt, 1)
      ELSE
        CALL spinteg(mass, mdot, spt, 1)
      ENDIF
      wkv = amax1(wkv, 0.e0)
      CALL spinteg(wkv, wdotkv, spt, 5)
      mass = mass + delt*mdot
c------------------------------ vehicle states module ------------------c
      CALL missil(t, massold, fxt, frcx, fyt, frcy, fzt, frcz, xyz,
     & xyzd, ud, vd, wd, gr, cim, xyzdd)
c---------------------------------------------------------------------c
c                              missile state integration module       c
c---------------------------------------------------------------------c
      CALL integ(xyzd(1), xyzdd(1), t, 6)
      CALL integ(xyzd(2), xyzdd(2), t, 7)
      CALL integ(xyzd(3), xyzdd(3), t, 8)
      CALL integ(xyz(1), xyzd(1), t, 9)
      CALL integ(xyz(2), xyzd(2), t, 10)
      CALL integ(xyz(3), xyzd(3), t, 11)
c  calculate current missile altitude
      alt = dsqrt(xyz(1)**2 + xyz(2)**2 + xyz(3)**2) - rade
        tmp_xyz(1) = xyz(1)
        tmp_xyz(2) = xyz(2)
        tmp_xyz(3) = xyz(3)
        CALL send_real_32bit(tmp_xyz(1))
        CALL send_real_32bit(tmp_xyz(2))
        CALL send_real_32bit(tmp_xyz(3))
      CALL send_real_32bit(alt)
c*********************************************************************
c                                                          *
c                        end of partition 1                          *
c                                                          *
c*********************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 30

      END
```

A.1.7 Boost2a1.for

```
      PROGRAM main
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION mvrwm, cir(9)
      DOUBLE PRECISION xyzr(3), xyz(3), xyzd(3), xyze(3)
      DOUBLE PRECISION cer(9), cri(9), cie(9)
      DOUBLE PRECISION cei(9), latlp, longlp
      DOUBLE PRECISION vrwi(3), cwr(9)
      DOUBLE PRECISION vrwind(3), viwind(3), vwwind(3)
      REAL fxa, fya, fza
      REAL mach, alfat, ca, cn, vrwm(3), qa
      REAL lat, long, rhod2, vsnd, cim(9)
      REAL shear, swdir, cwdir, vwind
#include "include/constant.dat"
      DATA omegae/0.0/, dtr/0.017453292519943296/
      DATA sref1/1.968953/, sref2/1.968953/
      DATA latlp, longlp, tmp1/3*0.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

c-------------------------------------------------------------------------c
c----------------------------- missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      CALL mmk(-90.0*dtr, 1, latlp*dtr, 2, longlp*dtr, 3, cei)
      CALL trans(cei, cie)
c-------------------------------------------------------------------------
c  initialize processor inputs if not already initialized
c p2
      qa = 0.0
      mach = 0.0
c  p5
      DO 10 i = 1, 3
        vrwm(i) = 0.0
   10 CONTINUE
c------------------------------ ----------------------------------------c
c------------------------------ main execution loop --------------------c
c-------------------------------------------------------------------------c
   20 CONTINUE
c*************************************************************************
c                                                      *
c                              partition 1                   *
c                                                      *
c*************************************************************************
      CALL send_real_32bit(mach)
      CALL send_real_32bit(qa)
      CALL send_real_32bit(vrwm(1))
      CALL send_real_32bit(vrwm(2))
      CALL send_real_32bit(vrwm(3))
      CALL mmk(0.0d0, 1, 0.0d0, 2, omegae*t, 3, cer)
      CALL mmlxy(cer, cie, cir)
      CALL trans(cir, cri)
c----------------- get parameters from main partition 1 thread ---------c
      CALL receive_real_64bit(xyz(1))
      CALL receive_real_64bit(xyz(2))
      CALL receive_real_64bit(xyz(3))
      CALL receive_real_64bit(xyzd(1))
      CALL receive_real_64bit(xyzd(2))
      CALL receive_real_64bit(xyzd(3))
      CALL receive_real_32bit(cim(1))
```

Appendix A
Exosim v1.0 Boost Phase

A.1 Mainline (FORTRAN)

A.1.1 Aeroca.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL ca1m(205), ca2m(205)
      REAL mach, alfat, ca
#include "include/constant.dat"
#include "include/ca1m.dat"
#include "include/ca2m.dat"
      DATA icam1/0/, icaa1/0/
      DATA icam2/0/, icaa2/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_net_32(mach)
      CALL receive_net_32(alfat)

      IF (t .GT. tstg1) THEN
c second stage
        CALL tlu2ei(mach, alfat, ca2m, icam2, icaa2, ca)
      ELSE
c first stage
        CALL tlu2ei(mach, alfat, ca1m, icam1, icaa1, ca)
      ENDIF

      CALL send_net_32(ca)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.2 Aerocn.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL cna1(205), cna2(205)
      REAL mach, alfat, cn
#include "include/constant.dat"
#include "include/cna1.dat"
#include "include/cna2.dat"
      DATA icnm1/0/, icna1/0/
      DATA icnm2/0/, icna2/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(mach)
      CALL receive_real_32bit(alfat)

      IF (t .GT. tstg1) THEN
c second stage
        CALL tlu2ei(mach, alfat, cna2, icnm2, icna2, cn)
      ELSE
c first stage
        CALL tlu2ei(mach, alfat, cna1, icnm1, icna1, cn)
      ENDIF

      CALL send_real_32bit(cn)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.3 Aeroxcp.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL xcpl1(205), xcpl2(205)
      REAL mach, alfat, xcp
#include "include/constant.dat"
#include "include/xcpl1.dat"
#include "include/xcpl2.dat"
      DATA ixcpm1/0/, ixcpa1/0/
      DATA ixcpm2/0/, ixcpa2/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(mach)
      CALL receive_real_32bit(alfat)

      IF (t .GT. tstg1) THEN
c second stage
        CALL tlu2ei(mach, alfat, xcpl2, ixcpm2, ixcpa2, xcp)
      ELSE
c first stage
        CALL tlu2ei(mach, alfat, xcpl1, ixcpm1, ixcpa1, xcp)
      ENDIF
      xcp =  - xcp/12.0

      CALL send_real_32bit(xcp)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.4 Attlm.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL attltt(5), attlmt(5)
      REAL attlm
#include "include/constant.dat"
#include "include/attltt.dat"
#include "include/attlmt.dat"
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL table(attltt, attlmt, t, attlm, 5, itable)

      CALL send_real_32bit(attlm)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.5 Bauto.for

```
      PROGRAM main
c-----------------------------------------------------------------------
c      subroutine bauto(t,thter,psier,sq,sr,ii,cgest,vrwm,alt,cmmd,
c     .                 dlpc,dlyc,mdltfr,malpha)
c-----------------------------------------------------------------------
c     function :            provides control of the missile about three
c                           axes throughout the boost phase of flight
c     inputs :              t,thter,psier,sq,sr,ii,cgest,vrwm,alt
c     outputs :             cmmd,dlpc,dlyc,mdltfr,malpha
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL cgest(3), ii(3), alt, cne, xcpe
      REAL estmch, alfate
      REAL cmmd(2)
      REAL ktht, kpsi, kthtd
      REAL kpsid, timte1(26), timte2(29)
      REAL thrte1(26), thrte2(29), altte(59)
      REAL rhote(59), vrwm(3)
      REAL ld, kne, kme
      REAL lfracs, frcloc(3, 4)
      REAL kthtk1, kthtk2
      REAL krtfrc, kthfrc
      REAL kthfm1, kthfm2, vsndte(59)
      REAL presste(59), mchlim, wmfrtt(4)
      REAL wmfrct(4)
      REAL dlpc, dlyc, sq, sr, mdltfr, malpha
#include "include/constant.dat"
#include "include/frcloc.dat"
#include "include/altte.dat"
#include "include/rhote.dat"
#include "include/presste.dat"
#include "include/vsndte.dat"
#include "include/timte1.dat"
#include "include/timte2.dat"
#include "include/thrte1.dat"
#include "include/thrte2.dat"
      DATA tapu/0.0/, dtapu/0.005/, tapustep/5.0/
      DATA mchlim/4.0/, kthtk1, kthtk2/.6, 1.5/
      DATA tign/0.01/, tst2on/22.995/, tfrcs/23.0/, tmode2/23.01/
      DATA wmtvc/25.0/, zettvc/0.85/, wmfrtt/0.0, 9.5, 39.95, 100./
      DATA wmfrct/62.83, 62.83, 42., 42./, zetfrc/0.85/, delon/0.045/
      DATA bcklmt/0.15/, delthg/0.045/, thjet/370./, sjet/1.3273/
      DATA sref1/1.968953/, sref2/1.968953/, aexite/.305/, aexit2/0.99/
      DATA xnoze/-12.5583/, xnoz2/-7.39107/, djet/1.3/, xjet/-2.71/
      DATA ialte/0/, ith1e/0/, ith2e/0/, iwmfrc/0/
      DATA dtr/0.017453292519943296/, slglbm/32.174048/

c initialize time
      tstep = 0.0
      t = tstep * delt

      cmmd(1) = 0.0
      cmmd(2) = 0.0
      dlpc = 0.0
      dlyc = 0.0
      mdltfr = 0.0
      malpha = 0.0
      sq = 0.0
      sr = 0.0
```

```
 10 CONTINUE
    CALL send_real_32bit(cmmd(1))
    CALL send_real_32bit(cmmd(2))
    CALL send_real_32bit(dlpc)
    CALL send_real_32bit(dlyc)
    CALL send_real_32bit(sq)
    CALL send_real_32bit(sr)
    CALL send_real_32bit(mdltfr)
    CALL send_real_32bit(malpha)
    CALL receive_real_32bit(cgest(1))
    CALL receive_real_32bit(cgest(2))
    CALL receive_real_32bit(cgest(3))
    CALL receive_real_32bit(ii(2))
    CALL receive_real_32bit(alt)
    CALL receive_real_32bit(vrwm(1))
    CALL receive_real_32bit(vrwm(2))
    CALL receive_real_32bit(vrwm(3))

    IF (tstep .GE. tapu) THEN
      tapu = tapu + tapustep

      IF (t .LT. tstg2) THEN
        IF (abs(t - tstg1) .LE. dteps) THEN
          aexite = aexit2
          xnoze = xnoz2
        ENDIF
        estalt = alt
        CALL table(altte, rhote, estalt, estrho, 59, ialte)
        estrho = estrho*1.0e-6/slglbm
        CALL table(altte, presste, estalt, estpre, 59, ialte)
        CALL table(altte, vsndte, estalt, estvsd, 59, ialte)
        estvel = sqrt(vrwm(1)**2 + vrwm(2)**2 + vrwm(3)**2)
        estmch = estvel/estvsd
        estqa = estrho*estvel**2/2.0
        IF (t .GT. tstg1) THEN
          srefe = sref2
          t0 = t - tst2on
          CALL table(timte2, thrte2, t0, thrve, 29, ith2e)
        ELSE
          t0 = t - tign
          srefe = sref1
          CALL table(timte1, thrte1, t0, thrve, 26, ith1e)
        ENDIF
        thre = thrve - aexite*estpre
        IF (thre .LT. 0.0) thre = 0.0
        IF (estvel .GT. 0.0) THEN
          alfate = sparctan(sqrt(vrwm(2)**2 + vrwm(3)**2), abs(vrwm
     &      (1)))/dtr
        ELSE
          alfate = 0.0
        ENDIF

        CALL send_real_32bit(estmch)
        CALL send_real_32bit(alfate)
*         if ( t.lt.tstg1 ) then
*            call tlu2ei(estmch,4.0d0,cna1e,icnm1e,icna1e,cne)
*            call tlu2ei(estmch,alfate,xcpl1e,icpm1e,icpa1e,xcpe)
*         else
*            call tlu2ei(estmch,4.0d0,cna2e,icnm2e,icna2e,cne)
*            call tlu2ei(estmch,alfate,xcpl2e,icpm2e,icpa2e,xcpe)
*         end if
        CALL receive_real_32bit(cne)
        CALL receive_real_32bit(xcpe)
```

```
c  conversion from inches to feet
          xcpe =  - xcpe/12.0
c  calculate cnalfa (per radian)
          cnalp = cne/(4.0*dtr)
          xcpcg = xcpe - cgest(1)
          IF (thre .GE. 1000.0 .AND. ii(2) .GT. 1.0e-6) malpha = abs
     &     (cnalp*xcpcg*srefe*estqa/ii(2))

          CALL receive_real_32bit(psier)
          CALL receive_real_32bit(thter)
          CALL receive_real_32bit(sq)
          CALL receive_real_32bit(sr)

c  tvc autopilot
          IF (t .LT. tmode2) THEN
            IF (thre .GE. 1000.0 .AND. ii(2) .GT. 1.0e-6) THEN
              xdel = cgest(1) - xnoze
              ktht = (ii(2)*wmtvc**2 + cnalp*srefe*estqa*xcpcg)/(thre
     &         *xdel)
              kpsi = (ii(2)*wmtvc**2 + cnalp*srefe*estqa*xcpcg)/(thre
     &         *xdel)
              kthtd = 2.0*zettvc*wmtvc*ii(2)/(thre*xdel)
              kpsid = 2.0*zettvc*wmtvc*ii(2)/(thre*xdel)
            ELSE
              ktht = 4.0
              kpsi = 4.0
              kthtd = 4.0
              kpsid = 4.0
            ENDIF
            cmmd(1) = thter*ktht - sq*kthtd
            cmmd(2) = psier*kpsi - sr*kpsid
            totcmd = sqrt(cmmd(1)**2 + cmmd(2)**2)
            IF (totcmd .GT. bcklmt) THEN
              cmmd(1) = cmmd(1)*bcklmt/totcmd
              cmmd(2) = cmmd(2)*bcklmt/totcmd
            ENDIF
          ELSE
            cmmd(1) = 0.0
            cmmd(2) = 0.0
          ENDIF
c  forward reaction control system autopilot
          IF (t .GE. tfrcs) THEN
            IF (thre .GE. 1000.0 .AND. ii(2) .GT. 1.0e-6) THEN
              ld = (xjet - xnoze)/djet
              ct = thjet/(estqa*sjet)
              IF (estmch .LE. mchlim) THEN
                kne = 0.6118 + (0.1358*(1. - 0.485*sqrt(ld))/sqrt(ct)
     &           + 0.0946*estmch + 0.004317/ld
              ELSE
                kne = 1.0 + exp(1.1 - 0.2116*(log(ct) + 8.5)**1.4)
              ENDIF
              kme = 0.5582 - 0.1884/sqrt(ct) - 1.9659/ld
              lfracs = frcloc(1, 1) - cgest(1)
              mdltfr = ( - kme*thjet*djet + kne*thjet*lfracs)/ii(2)
              CALL table(wmfrtt, wmfrct, t - tmode2, wmfrc, 4,
     &         iwmfrc)
              krtfrc = 2.0*zetfrc*wmfrc/(wmfrc**2 + malpha)
              kthfrc = 2.0*delon/(mdltfr*krtfrc*dtapu)
              kthfm1 = delon*malpha/mdltfr
              kthfm2 = delon/delthg
              IF (kthfrc .LT. kthfm1) kthfrc = kthfm1
              IF (kthfrc .LT. kthfm2) kthfrc = kthfm2
              ktht = kthfrc*kthtk1
              kthtd = ktht*krtfrc*kthtk2
```

```
              kpsi = ktht
              kpsid = kthtd
            ELSE
              malpha = 544.18
              mdltfr = 6.0437
              ktht = 10.0
              kthtd = 25.0
              kpsi = 10.0
              kpsid = 25.0
            ENDIF
            dlpc = thter*ktht - sq*kthtd
            dlyc = psier*kpsi - sr*kpsid
          ENDIF
        ELSE
          CALL send_real_32bit(estmch)
          CALL send_real_32bit(alfate)
          CALL receive_real_32bit(cne)
          CALL receive_real_32bit(xcpe)
          CALL receive_real_32bit(psier)
          CALL receive_real_32bit(thter)
          CALL receive_real_32bit(sq)
          CALL receive_real_32bit(sr)
        ENDIF
      ELSE
        CALL send_real_32bit(estmch)
        CALL send_real_32bit(alfate)
        CALL receive_real_32bit(cne)
        CALL receive_real_32bit(xcpe)
        CALL receive_real_32bit(psier)
        CALL receive_real_32bit(thter)
        CALL receive_real_32bit(sq)
        CALL receive_real_32bit(sr)
      ENDIF

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.6 Boost2a.for

```
      PROGRAM main
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      REAL gr(3)
      DOUBLE PRECISION latlp, longlp
      DOUBLE PRECISION xyz(3)
      DOUBLE PRECISION xyzd(3), xyzdd(3), xyze(3)
      DOUBLE PRECISION xyzed(3), xyzedd(3), cei(9)
      REAL mdotf, msstg2, mass, massold, mass0
      REAL frcx, frcy, frcz, alt
      REAL mdot, fxt, fyt, fzt
      REAL ud, vd, wd, wkv, wkv0, wdotkv
      REAL cim(9), mdott, spt
        REAL tmp_xyz(3)
#include "include/constant.dat"
      DATA rade/20898908.0/, msstg2/19.457/
      DATA dtr/0.017453292519943296/, slglbm/32.174048/
      DATA mass0/43.939/, wkv0/97.1/
      DATA latlp/0.0/, longlp/0.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

      DO 10 i = 1, 3
        xyze(i) = 0.0
        xyzed(i) = 0.0
        xyzedd(i) = 0.0
   10 CONTINUE
      xyze(1) = rade
      alt = dsqrt(xyze(1)**2 + xyze(2)**2 + xyze(3)**2) - rade
c-------------------------------------------------------------------------c
c------------------------------ missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      mass = mass0
      wkv = wkv0
      CALL mmk(-90.0*dtr, 1, latlp*dtr, 2, longlp*dtr, 3, cei)
      CALL vecrot(xyzed, cei, xyzd)
      CALL vecrot(xyze, cei, xyz)
      mdot = 0.0
      wdotkv = 0.0
      CALL vecrot(xyzedd, cei, xyzdd)
      spt = t
      CALL spintegi(mass, mdot, spt, 1)
      CALL spintegi(wkv, wdotkv, spt, 5)
      CALL integi(xyzd(1), xyzdd(1), t, 6)
      CALL integi(xyzd(2), xyzdd(2), t, 7)
      CALL integi(xyzd(3), xyzdd(3), t, 8)
      CALL integi(xyz(1), xyzd(1), t, 9)
      CALL integi(xyz(2), xyzd(2), t, 10)
      CALL integi(xyz(3), xyzd(3), t, 11)
c-------------------------------------------------------------------------
c  initialize processor inputs if not already initialized
c pl
      fxt = 0.0
      fyt = 0.0
      fzt = 0.0
      mdott = 0.0
      frcx = 0.0
      frcy = 0.0
      frcz = 0.0
```

```
      mdotf = 0.0
c p2
c  p3
      ud = 0.0
      vd = 0.0
      wd = 0.0
c  p4
      DO 20 i = 1, 3
        gr(i) = 0.0
   20 CONTINUE
c  p5
c  initialization routine
      mass = mass + delt*mdot
c-------------------------------------------------------------------------c
c------------------------------ main execution loop ----------------------c
c-------------------------------------------------------------------------c
   30 CONTINUE
c*************************************************************************
c                                                        *
c                              partition 1                    *
c                                                        *
c*************************************************************************
c--------------------------- missile state update module ----------------c
c  temporarily extrapolate missile states from last integration
c  step ( note : the extrapolated states are overwritten when
c                the true integration is performed )
c-------------- send parameters to partitions 3, 4, and 5 ---------------c
      CALL send_real_32bit(ud)
      CALL send_real_32bit(vd)
      CALL send_real_32bit(wd)
      CALL send_real_32bit(gr(1))
      CALL send_real_32bit(gr(2))
      CALL send_real_32bit(gr(3))
      CALL send_real_32bit(alt)
c---------- send mass to masspr subroutine table lookup processors ----c
      CALL send_real_32bit(mass)
      xyzd(1) = xyzd(1) + delt*xyzdd(1)
      xyzd(2) = xyzd(2) + delt*xyzdd(2)
      xyzd(3) = xyzd(3) + delt*xyzdd(3)
      xyz(1) = xyz(1) + delt*xyzd(1)
      xyz(2) = xyz(2) + delt*xyzd(2)
      xyz(3) = xyz(3) + delt*xyzd(3)
c  calculate current missile altitude
      alt = dsqrt(xyz(1)**2 + xyz(2)**2 + xyz(3)**2) - rade
c---------- send altitude to atmos subroutine table lookup processors -c
      CALL send_real_32bit(alt)
c---------- send parameters to thread containing winds subroutine -----c
      CALL send_real_64bit(xyz(1))
      CALL send_real_64bit(xyz(2))
      CALL send_real_64bit(xyz(3))
      CALL send_real_64bit(xyzd(1))
      CALL send_real_64bit(xyzd(2))
      CALL send_real_64bit(xyzd(3))
      CALL receive_real_32bit(cim(1))
      CALL receive_real_32bit(cim(2))
      CALL receive_real_32bit(cim(3))
      CALL receive_real_32bit(cim(4))
      CALL receive_real_32bit(cim(5))
      CALL receive_real_32bit(cim(6))
      CALL receive_real_32bit(cim(7))
      CALL receive_real_32bit(cim(8))
      CALL receive_real_32bit(cim(9))
c---------- receive parameters from partition #2 ---------------------c
      CALL receive_real_32bit(fxt)
```

```
      CALL receive_real_32bit(fyt)
      CALL receive_real_32bit(fzt)
      CALL receive_real_32bit(frcx)
      CALL receive_real_32bit(frcy)
      CALL receive_real_32bit(frcz)
      CALL receive_real_32bit(mdott)
      CALL receive_real_32bit(mdotf)
      wkv = wkv + delt*wdotkv
      mdotkv =  - mdotf*slglbm
      mdot =  - mdott - mdotf
c  save mass value for use in missil subroutine
      massold = mass
      spt = t
c  trapezoidal integration for simplicity
      IF (dabs(t - tstg1) .LE. dteps) THEN
c  first stage separation
        mass = msstg2
        CALL spintegi(mass, 0.0e0, spt, 1)
      ELSEIF (dabs(t - tstg2) .LE. dteps) THEN
c  second stage separation
        mass = wkv/slglbm
        CALL spintegi(mass, 0.0e0, spt, 1)
      ELSE
        CALL spinteg(mass, mdot, spt, 1)
      ENDIF
      wkv = amax1(wkv, 0.e0)
      CALL spinteg(wkv, wdotkv, spt, 5)
      mass = mass + delt*mdot
c------------------------------ vehicle states module ------------------c
      CALL missil(t, massold, fxt, frcx, fyt, frcy, fzt, frcz, xyz,
     & xyzd, ud, vd, wd, gr, cim, xyzdd)
c-----------------------------------------------------------------------c
c                              missile state integration module         c
c-----------------------------------------------------------------------c
      CALL integ(xyzd(1), xyzdd(1), t, 6)
      CALL integ(xyzd(2), xyzdd(2), t, 7)
      CALL integ(xyzd(3), xyzdd(3), t, 8)
      CALL integ(xyz(1), xyzd(1), t, 9)
      CALL integ(xyz(2), xyzd(2), t, 10)
      CALL integ(xyz(3), xyzd(3), t, 11)
c  calculate current missile altitude
      alt = dsqrt(xyz(1)**2 + xyz(2)**2 + xyz(3)**2) - rade
        tmp_xyz(1) = xyz(1)
        tmp_xyz(2) = xyz(2)
        tmp_xyz(3) = xyz(3)
        CALL send_real_32bit(tmp_xyz(1))
        CALL send_real_32bit(tmp_xyz(2))
        CALL send_real_32bit(tmp_xyz(3))
      CALL send_real_32bit(alt)
c*************************************************************************
c                                                                       *
c                        end of partition 1                                *
c                                                                       *
c*************************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 30

      END
```

A.1.7 Boost2a1.for

```
      PROGRAM main
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION mvrwm, cir(9)
      DOUBLE PRECISION xyzr(3), xyz(3), xyzd(3), xyze(3)
      DOUBLE PRECISION cer(9), cri(9), cie(9)
      DOUBLE PRECISION cei(9), latlp, longlp
      DOUBLE PRECISION vrwi(3), cwr(9)
      DOUBLE PRECISION vrwind(3), viwind(3), vwwind(3)
      REAL fxa, fya, fza
      REAL mach, alfat, ca, cn, vrwm(3), qa
      REAL lat, long, rhod2, vsnd, cim(9)
      REAL shear, swdir, cwdir, vwind
#include "include/constant.dat"
      DATA omegae/0.0/, dtr/0.017453292519943296/
      DATA sref1/1.968953/, sref2/1.968953/
      DATA latlp, longlp, tmp1/3*0.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

c-------------------------------------------------------------------------c
c----------------------------- missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      CALL mmk(-90.0*dtr, 1, latlp*dtr, 2, longlp*dtr, 3, cei)
      CALL trans(cei, cie)
c-------------------------------------------------------------------------
c  initialize processor inputs if not already initialized
c p2
      qa = 0.0
      mach = 0.0
c  p5
      DO 10 i = 1, 3
        vrwm(i) = 0.0
   10 CONTINUE
c-------------------------------------------------------------------------c
c----------------------------- main execution loop --------------------c
c-------------------------------------------------------------------------c
   20 CONTINUE
c*************************************************************************
c                                                        *
c                            partition 1                      *
c                                                        *
c*************************************************************************
      CALL send_real_32bit(mach)
      CALL send_real_32bit(qa)
      CALL send_real_32bit(vrwm(1))
      CALL send_real_32bit(vrwm(2))
      CALL send_real_32bit(vrwm(3))
      CALL mmk(0.0d0, 1, 0.0d0, 2, omegae*t, 3, cer)
      CALL mmlxy(cer, cie, cir)
      CALL trans(cir, cri)
c---------------- get parameters from main partition 1 thread ---------c
      CALL receive_real_64bit(xyz(1))
      CALL receive_real_64bit(xyz(2))
      CALL receive_real_64bit(xyz(3))
      CALL receive_real_64bit(xyzd(1))
      CALL receive_real_64bit(xyzd(2))
      CALL receive_real_64bit(xyzd(3))
      CALL receive_real_32bit(cim(1))
```

```
      CALL receive_real_32bit(cim(2))
      CALL receive_real_32bit(cim(3))
      CALL receive_real_32bit(cim(4))
      CALL receive_real_32bit(cim(5))
      CALL receive_real_32bit(cim(6))
      CALL receive_real_32bit(cim(7))
      CALL receive_real_32bit(cim(8))
      CALL receive_real_32bit(cim(9))
      xyze(1) = cie(1)*xyz(1) + cie(4)*xyz(2) + cie(7)*xyz(3)
      xyze(2) = cie(2)*xyz(1) + cie(5)*xyz(2) + cie(8)*xyz(3)
      xyze(3) = cie(3)*xyz(1) + cie(6)*xyz(2) + cie(9)*xyz(3)
      CALL vecrot(xyze, cer, xyzr)
c  calculate current latitude and longitude
      lat = arctan(xyzr(3), dsqrt(xyzr(1)**2 + xyzr(2)**2))
      long = arctan(xyzr(2), xyzr(1))
c**********   start of winds subroutine ******************************c
c      call mmk(0.0d0,1,-lat,2,long,3,crw)
c      call trans(crw,cwr)
      a = cos( - lat)
      b = sin( - lat)
      c = cos(long)
      d = sin(long)
      cwr(1) = a*c
      cwr(2) = d
      cwr(3) = b*c
      cwr(4) = a*d
      cwr(5) = c
      cwr(6) = b*d
      cwr(7) = b
      cwr(8) = 0.0
      cwr(9) = a
c----------- get masspr table look up values from other processors ----c
      CALL receive_real_32bit(vwind)
      CALL receive_real_32bit(shear)
      CALL receive_real_32bit(swdir)
      CALL receive_real_32bit(cwdir)
:      call vmk(shear,cwdir*vwind,swdir*vwind,vwwind)
      vwwind(1) = shear
      vwwind(2) = cwdir*vwind
      vwwind(3) = swdir*vwind
      CALL vecrot(vwwind, cwr, vrwind)
      CALL vecrot(vrwind, cri, viwind)
      CALL vecsub(xyzd, viwind, vrwi)
c      call vecrot(vrwi,cim,vrwm)
      vrwm(1) = cim(1)*vrwi(1) + cim(4)*vrwi(2) + cim(7)*vrwi(3)
      vrwm(2) = cim(2)*vrwi(1) + cim(5)*vrwi(2) + cim(8)*vrwi(3)
      vrwm(3) = cim(3)*vrwi(1) + cim(6)*vrwi(2) + cim(9)*vrwi(3)
      mvrwm = sqrt(vrwm(1)**2 + vrwm(2)**2 + vrwm(3)**2)
c**********  end of winds subroutine *********************************c
c------------ calculate parameters from aero and start table lookups -c
      CALL receive_real_32bit(vsnd)
      mach = mvrwm/vsnd
      tmp1 = sqrt(vrwm(2)**2 + vrwm(3)**2)
      tmp2 = abs(vrwm(1))
      alfat = arctan(tmp1, tmp2)/dtr
      CALL send_real_32bit(mach)
      CALL send_real_32bit(alfat)
      CALL receive_real_32bit(rhod2)
      qa = (mvrwm**2)*rhod2
c-------------perform part of aero subroutine -------------------------c
      IF (mvrwm .LE. 0.0 .OR. t .GE. tstg2) THEN
        fxa = 0.0e0
        fya = 0.0e0
        fza = 0.0e0
```

```
          CALL receive_real_32bit(ca)
          CALL receive_real_32bit(cn)
        ELSE
          IF (dabs(tmp1) .GT. 1.0d-6) THEN
            cphia = vrwm(3)/tmp1
            sphia = vrwm(2)/tmp1
          ELSE
            cphia = 1.0d0
            sphia = 0.0d0
          ENDIF
          IF (t .GT. tstg1) THEN
            sur = sref2
          ELSE
            sur = sref1
          ENDIF
          qs = qa*sur
          CALL receive_real_32bit(ca)
          CALL receive_real_32bit(cn)
          fxa = qs*ca
          fya =  - qs*cn*sphia
          fza =  - qs*cn*cphia
        ENDIF
        CALL send_real_32bit(fxa)
        CALL send_real_32bit(fya)
        CALL send_real_32bit(fza)
c***********************************************************************
c                                                                      *
c                        end of partition 1                                *
c                                                                      *
c***********************************************************************
c increment time
        tstep = tstep + 1.0
        t = tstep * delt

        IF (t  LT. tfinal) GOTO 20

        END
```

A.1.8 Boost2a2.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL cim(9)
      REAL quat(4), quatd(4)
      REAL cmi(9), pqr(3)
      REAL pd, qd, rd
      REAL told
#include "include/constant.dat"
      DATA dtr/0.017453292519943296/
      DATA tmp1/0.0/, quatm/1.0/
      DATA phiicd/0.0/, thticd/-35.0/, psiicd/0.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

      DO 10 i = 1, 3
        pqr(i) = 0.0
   10 CONTINUE
      phi = phiicd*dtr
      tht = thticd*dtr
      psi = psiicd*dtr
      CALL spmmk(phi, 1, tht, 2, psi, 3, cim)
      CALL sptrans(cim, cmi)
c-------------------------------------------------------------------------c
c------------------------------ missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      CALL bxi2fv(quatm, cmi, quat)
      pd = 0.0
      qd = 0.0
      rd = 0.0
      CALL fvdot(pqr, tmp1, quat, quatd)
      CALL spintegi(pqr(1), pd, t, 12)
      CALL spintegi(pqr(2), qd, t, 13)
      CALL spintegi(pqr(3), rd, t, 14)
      CALL spintegi(quat(1), quatd(1), t, 15)
      CALL spintegi(quat(2), quatd(2), t, 16)
      CALL spintegi(quat(3), quatd(3), t, 17)
      CALL spintegi(quat(4), quatd(4), t, 18)
      p = pqr(1)
      q = pqr(2)
      r = pqr(3)
      itst = 0
c------------------------------------------------ -------------------c
c------------------------------ main execution loop -----------------c
c-------------------------------------------------------------------c
   20 CONTINUE
c*******************************************************************
c                                                      *
c                            partition 1                    *
c                                                      *
c*******************************************************************
      CALL send_real_32bit(cim(1))
      CALL send_real_32bit(cim(2))
      CALL send_real_32bit(cim(3))
      CALL send_real_32bit(cim(4))
      CALL send_real_32bit(cim(5))
      CALL send_real_32bit(cim(6))
      CALL send_real_32bit(cim(7))
      CALL send_real_32bit(cim(8))
```

```
      CALL send_real_32bit(cim(9))

C       Added for graphics program
        CALL send_real_32bit(phi)
        CALL send_real_32bit(tht)
        CALL send_real_32bit(psi)

      CALL receive_real_32bit(pd)
      CALL receive_real_32bit(qd)
      CALL receive_real_32bit(rd)
      IF (itst .EQ. 0) THEN
        itst = 1
      ELSE
        CALL spinteg(p, pd, told, 12)
        CALL spinteg(q, qd, told, 13)
        CALL spinteg(r, rd, told, 14)
      ENDIF
      p = p + delt*pd
      q = q + delt*qd
      r = r + delt*rd
      quat(1) = quat(1) + delt*quatd(1)
      quat(2) = quat(2) + delt*quatd(2)
      quat(3) = quat(3) + delt*quatd(3)
      quat(4) = quat(4) + delt*quatd(4)
c---------section of missil subroutine that finds phi, tht, and psi----c
      call vmk(p,q,r,pqr)
      tmp2 = 0.0
      CALL fvdot(pqr, tmp2, quat, quatd)
      CALL fv2bxi(quat, tmp2, cmi)
      CALL sptrans(cmi, cim)
      phi = sparctan(cim(8), cim(9))
      tht =  - arcsin(cim(7))
      psi = sparctan(cim(4), cim(1))
c-------------------------------------------------------------------------c
c                              missile state integration module           c
c-------------------------------------------------------------------------c
      CALL spinteg(quat(1), quatd(1), t, 15)
      CALL spinteg(quat(2), quatd(2), t, 16)
      CALL spinteg(quat(3), quatd(3), t, 17)
      CALL spinteg(quat(4), quatd(4), t, 18)
c*************************************************************************
c                                                    *
c                         end of partition 1                          *
c                                                    *
c*************************************************************************
      told = t

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 20

      END
```

A.1.9 Boost2a3.for

```
      PROGRAM main
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
c $include(':pfp:include/target.for')
      DOUBLE PRECISION mxa, mya, mza
      DOUBLE PRECISION xyzlch(3)
      DOUBLE PRECISION xyz(3), gb(3)
      DOUBLE PRECISION uxyz(3), mx, my, mz
      DOUBLE PRECISION gr(3), mgr, mxyz
      REAL fxa, fya, fza
      REAL cim(9), cmi(9)
      REAL ixx, iyy, izz, mass, cg(3), xcp
      REAL mrcx, mrcy, mrcz, frcx, frcy, frcz
      REAL fxt, fyt, fzt, mxt, myt, mzt
      REAL p, q, r, pd, qd, rd
      REAL spt, pqr(3)
#include "include/constant.dat"
      DATA rade/20898908.0/
      DATA nclear/0/, imis/0/, xlnch/3.0/
      DATA gmu/1.4052477e16/

c initialize time
      tstep = 0.0
      t = tstep * delt

      DO 10 i = 1, 3
        pqr(i) = 0.0
   10 CONTINUE
c-------------------------------------------------------------------------c
c------------------------------ missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      pd = 0.0
      qd = 0.0
      rd = 0.0
      spt = t
      CALL spintegi(pqr(1), pd, spt, 12)
      CALL spintegi(pqr(2), qd, spt, 13)
      CALL spintegi(pqr(3), rd, spt, 14)
      p = pqr(1)
      q = pqr(2)
      r = pqr(3)
      DO 20 i = 1, 3
        gr(i) = 0.0
   20 CONTINUE
c-------------------------------------------------------------------------c
c------------------------------ main execution loop ----------------------c
c-------------------------------------------------------------------------c
   30 CONTINUE
c*************************************************************************
c                                                        *
c                            partition 1                      *
c                                                        *
c*************************************************************************
c-------------- send parameters to partitions 3, 4, and 5 --------------c
      CALL send_real_32bit(p)
      CALL send_real_32bit(q)
      CALL send_real_32bit(r)
      CALL send_real_32bit(pd)
      CALL send_real_32bit(qd)
      CALL send_real_32bit(rd)
      CALL receive_real_32bit(mass)
```

```
      CALL receive_real_64bit(xyz(1))
      CALL receive_real_64bit(xyz(2))
      CALL receive_real_64bit(xyz(3))
      CALL receive_real_32bit(cim(1))
      CALL receive_real_32bit(cim(2))
      CALL receive_real_32bit(cim(3))
      CALL receive_real_32bit(cim(4))
      CALL receive_real_32bit(cim(5))
      CALL receive_real_32bit(cim(6))
      CALL receive_real_32bit(cim(7))
      CALL receive_real_32bit(cim(8))
      CALL receive_real_32bit(cim(9))
c---------- receive parameters from partition #2 ----------------------c
      CALL receive_real_32bit(fxt)
      CALL receive_real_32bit(fyt)
      CALL receive_real_32bit(fzt)
      CALL receive_real_32bit(mxt)
      CALL receive_real_32bit(myt)
      CALL receive_real_32bit(mzt)
      CALL receive_real_32bit(frcx)
      CALL receive_real_32bit(frcy)
      CALL receive_real_32bit(frcz)
      CALL receive_real_32bit(mrcx)
      CALL receive_real_32bit(mrcy)
      CALL receive_real_32bit(mrcz)
c---------------------------- mass properties module ------------------c
      CALL receive_real_32bit(cg(1))
      CALL receive_real_32bit(cg(2))
      CALL receive_real_32bit(cg(3))
      CALL receive_real_32bit(ixx)
      CALL receive_real_32bit(iyy)
      CALL receive_real_32bit(izz)
      p = p + delt*pd
      q = q + delt*qd
      r = r + delt*rd
c---------------------------- vehicle states module -----------------c
c--------------------------------------------------------------------
      IF (imis .EQ. 0) THEN
        CALL sptrans(cim, cmi)
        xyzlch(1) = xlnch*cmi(1) + rade
        xyzlch(2) = xlnch*cmi(2)
        xyzlch(3) = xlnch*cmi(3)
        imis = 1
      ENDIF
      CALL magt(xyz, mxyz, uxyz)
      mgr = gmu/mxyz**2
      CALL mvbys(-mgr, uxyz, gr)
c           CALL vecrot(gr, cim, gb)
      gb(1) = cim(1)*gr(1) + cim(4)*gr(2) + cim(7)*gr(3)
      gb(2) = cim(2)*gr(1) + cim(5)*gr(2) + cim(8)*gr(3)
      gb(3) = cim(3)*gr(1) + cim(6)*gr(2) + cim(9)*gr(3)

c----------------- section of aero subroutine to --------------------c
c----------------- calculate mxa,mya, and mza from fxa,fya,and fza --c
      CALL receive_real_32bit(fxa)
      CALL receive_real_32bit(fya)
      CALL receive_real_32bit(fza)
      CALL receive_real_32bit(xcp)
      mxa = fya*cg(3) - fza*cg(2)
      mya =  - fxa*cg(3) + fza*(cg(1) - xcp)
      mza = fxa*cg(2) - fya*(cg(1) - xcp)
c---------------- section of missil subroutine to find ---------------c
c----------------  pd, qd, and rd                        ---------------c
      fx = fxt + fxa + frcx
```

```
      fy = fyt + fya + frcy
      fz = fzt + fza + frcz
      mx = mxa + mxt + mrcx
      my = mya + myt + mrcy
      mz = mza + mzt + mrcz
      IF (nclear .EQ. 1) THEN
        pd = 0.0
        qd = my/iyy + r*p*((izz - ixx)/iyy)
        rd = mz/izz + p*q*((ixx - iyy)/izz)
      ELSEIF (fx/mass .LE. dabs(gb(1))) THEN
        pd = 0.0
        qd = 0.0
        rd = 0.0
      ELSEIF (xyz(1) .LE. xyzlch(1) .AND. xyz(2) .LE. xyzlch(2) .AND.
     & xyz(3) .LE. xyzlch(3)) THEN
        pd = 0.0
        qd = 0.0
        rd = 0.0
      ELSE
        nclear = 1
c        call output_message( %val(character_08bit),
c    .    ' missile has cleared the launcher' )
c        call output_nl
        pd = 0.0
        qd = my/iyy + r*p*((izz - ixx)/iyy)
        rd = mz/izz + p*q*((ixx - iyy)/izz)
      ENDIF
c-------------------------------------------------------------------------c
c                              missile state integration module           c
c-------------------------------------------------------------------------c
      spt = t
      CALL spinteg(p, pd, spt, 12)
      CALL spinteg(q, qd, spt, 13)
      CALL spinteg(r, rd, spt, 14)
c*************************************************************************
c                                                        *
c                        end of partition 1                                *
c                                                        *
c*************************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 30

      END
```

A.1.10 Boost2b.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL tmf(10, 4), thf(10, 4)
      INTEGER lenf(4)
      REAL mrcx, mrcy, mrcz
      REAL malpha, pm(3), mdotf
      REAL cg(3), mach
      REAL mdltfr
#include "include/constant.dat"
      DATA tfrac/23001.0/, dtfru/5.0/,tfrcs/23.001/

c initialize time
      tstep = 0.0
      t = tstep * delt

      DO 20 i = 1, 10
        DO 10 j = 1, 4
          tmf(i, j) = 0.0
          thf(i, j) = 0.0
   10   CONTINUE
   20 CONTINUE
      DO 30 i = 1, 4
        lenf(i) = 0
   30 CONTINUE
c-------------------------------------------------------------------------c
c------------------------------ missile state initialization module ----c
c p1
      frcx = 0.0
      frcy = 0.0
      frcz = 0.0
      mrcx = 0.0
      mrcy = 0.0
      mrcz = 0.0
      mdotf = 0.0
c-------------------------------------------------------------------------c
c------------------------------ main execution loop ----------------------c
c-------------------------------------------------------------------------c
   40 CONTINUE
c*************************************************************************
c                                                        *
c                            partition 2                      *
c                                                        *
c*************************************************************************
c---------------- send parameters to partition #1 ----------------------c
      CALL send_real_32bit(frcx)
      CALL send_real_32bit(frcy)
      CALL send_real_32bit(frcz)
      CALL send_real_32bit(mrcx)
      CALL send_real_32bit(mrcy)
      CALL send_real_32bit(mrcz)
      CALL send_real_32bit(mdotf)
c---------- receive parameters from partition #1 ----------------------c
      CALL receive_real_32bit(cg(1))
      CALL receive_real_32bit(cg(2))
      CALL receive_real_32bit(cg(3))
c---------- receive parameters from partition #1 ----------------------c
      CALL receive_real_32bit(mach)
      CALL receive_real_32bit(qa)
c---------- receive parameters from partition #3,4, and 5 -------------c
      CALL receive_real_32bit(dlpc)
      CALL receive_real_32bit(dlyc)
```

```
      CALL receive_real_32bit(sq)
      CALL receive_real_32bit(sr)
      CALL receive_real_32bit(mdltfr)
      CALL receive_real_32bit(malpha)
      CALL receive_real_32bit(pm(1))
      CALL receive_real_32bit(pm(2))
      CALL receive_real_32bit(pm(3))
c---------------------------- fracs thruster response module ---------c
      IF (t .GE. tfrcs .AND. t .LT. tstg2) THEN
        CALL frcthr(t, cg, mach, qa, tmf, thf, lenf, frcx, frcy, frcz,
     &   mrcx, mrcy, mrcz, mdotf)
c---------------------------- fracs logic module ----------------------c
        IF (tstep .GE. tfrac) THEN
          CALL fracs(t, dlpc, dlyc, sq, sr, mdltfr, malpha, pm, tmf,
     &     thf, lenf)
          tfrac = tfrac + dtfru
        ENDIF
      ELSE
        frcx = 0.
        frcy = 0.
        frcz = 0.
        mrcx = 0.
        mrcy = 0.
        mrcz = 0.
        mdotf = 0.
      ENDIF
c*************************************************************************
c                                                          *
c                        end of partition 2                              *
c                                                          *
c*************************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 40

      END
```

A.1.11 Boost2b1.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL mxt, myt, mzt, mdott
      REAL cmmd(2)
      REAL cg(3), press
#include "include/constant.dat"
      DATA tinhib/.35/
      DATA pmax/0.13963/
      DATA dlpic/0.0/, dlyic/0.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

c-------------------------------------------------------------------------c
c----------------------------- missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      dlp = dlpic
      dly = dlyic
      dlpd = 0.0
      dlyd = 0.0
      CALL spintegi(dlp, dlpd, t, 19)
      CALL spintegi(dly, dlyd, t, 20)
c-------------------------------------------------------------------------
c  initialize processor inputs if not already initialized
c p1
      fxt = 0.0
      fyt = 0.0
      fzt = 0.0
      mxt = 0.0
      myt = 0.0
      mzt = 0.0
      mdott = 0.0
c-------------------------------------------------------------------------c
c------------------------------ main execution loop --------------------c
c-------------------------------------------------------------------------c
   10 CONTINUE
c**********************.*************************************************
c                                                      *
c                            partition 2                     *
c                                                      *
c*****************************************************.*********************
c---------------- send parameters to partition #1 ----------------------c
      CALL send_real_32bit(fxt)
      CALL send_real_32bit(fyt)
      CALL send_real_32bit(fzt)
      CALL send_real_32bit(mxt)
      CALL send_real_32bit(myt)
      CALL send_real_32bit(mzt)
      CALL send_real_32bit(mdott)
c---------- receive parameters from partition #1 ----------------------c
      CALL receive_real_32bit(cg(1))
      CALL receive_real_32bit(cg(2))
      CALL receive_real_32bit(cg(3))
c---------- receive parameters from partition #1 ----------------------c
      CALL receive_real_32bit(press)
c---------- receive parameters from partition #3,4, and 5 -------------c
      CALL receive_real_32bit(cmmd(1))
      CALL receive_real_32bit(cmmd(2))
      IF (t .LE. tstg1) THEN
```

```
        dlp = dlp + delt*dlpd
        dly = dly + delt*dlyd
        totdel = sqrt(dlp**2 + dly**2)
        IF (totdel .GT. pmax) THEN
          dlp = dlp*pmax/totdel
          dly = dly*pmax/totdel
        ENDIF
      ENDIF
c------------------------------ boosters module ------------------------c
      CALL bthrst(t, cg, press, dlp, dly, fxt, fyt, fzt, mxt, myt, mzt,
     & mdott)
c------------------------------ nozzle control unit module -------------c
      IF (t .LE. tstg1) THEN
        IF (t .GT. tinhib) THEN
          CALL ncu(dlp, dly, cmmd, dlpd, dlyd)
        ELSE
          dlpd = 0.0
          dlyd = 0.0
        ENDIF
        CALL spinteg(dlp, dlpd, t, 19)
        CALL spinteg(dly, dlyd, t, 20)
        totdel = sqrt(dlp**2 + dly**2)
        IF (totdel .GT. pmax) THEN
          dlp = dlp*pmax/totdel
          dly = dly*pmax/totdel
        ENDIF
      ELSE
        dlp = 0.0
        dly = 0.0
      ENDIF
c*********************************************************************
c                                                          *
c                        end of partition 2                          *
c                                                          *
c*********************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.12 Boost2c.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL us(3), ac(3), vw(3), pg(3)
      REAL ti2m(9), at(3)
      REAL mvs, mvr, uvs(3), gr(3)
      REAL pqr(3)
      REAL usd(3), pgd(3), vwd(3)
      REAL mvrdot
      REAL pm(3), sq, sr, psier, thter, attlm
      REAL delphi, deltht, delpsi, delu, delv, delw
#include "include/constant.dat"
      DATA tst2on/22.995/
      DATA dtbgu/5.0/, epsl/2.5e-04/
      DATA dtr/0.017453292519943296/
      DATA thticd/-35.0/, psiicd/0.0/
      DATA vp1/13770.0/, us0d/-22.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

      DO 10 i = 1, 3
        pqr(i) = 0.0
        usd(i) = 0.0
        pgd(i) = 0.0
        vwd(i) = 0.0
        vw(i) = 0.0
   10 CONTINUE
      mvrdot = 0.0
      stht = thticd*dtr
      spsi = psiicd*dtr
      sq = pqr(2)
      sr = pqr(3)
      us(1) = cos(spsi)*cos(us0d*dtr)
      us(2) = sin(spsi)*cos(us0d*dtr)
      us(3) =  - sin(us0d*dtr)
      pg(1) = cos(spsi)*cos(stht)
      pg(2) = sin(spsi)*cos(stht)
      pg(3) =  - sin(stht)
c-------------------------------------------------------------------------c
c----------------------------- missile state initialization module -----c
c-------------------------------------------------------------------------
c  initialize processor inputs if not already initialized
      DO 20 i = 1, 3
        pm(i) = 0.0
   20 CONTINUE
      delu = 0.0
      delv = 0.0
      delw = 0.0
      DO 30 i = 1, 3
        at(i) = 0.0
   30 CONTINUE
      DO 40 i = 1, 9
        ti2m(i) = 0.0
   40 CONTINUE
      mvr = vp1
c  initialization routine
      tgpu = 0.0
      tlgpu = 0.0
c-------------------------------------------------------------------------c
c----------------------------- main execution loop --------------------c
```

```
c-----------------------------------------------------------------------c
   50 CONTINUE
c-------------- send parameters to partition #2 ----------------------c
      CALL send_real_32bit(at(1))
      CALL send_real_32bit(at(2))
      CALL send_real_32bit(at(3))
      CALL send_real_32bit(delxd)
      CALL send_real_32bit(delyd)
      CALL send_real_32bit(delzd)
      CALL send_real_32bit(pm(1))
      CALL send_real_32bit(pm(2))
      CALL send_real_32bit(pm(3))
c-------------- receive parameters from partition #1 ------------------c
      CALL receive_real_32bit(gr(1))
      CALL receive_real_32bit(gr(2))
      CALL receive_real_32bit(gr(3))
      CALL receive_real_32bit(mvs)
      CALL receive_real_32bit(uvs(1))
      CALL receive_real_32bit(uvs(2))
      CALL receive_real_32bit(uvs(3))
      CALL receive_real_32bit(delphi)
      CALL receive_real_32bit(deltht)
      CALL receive_real_32bit(delpsi)
      CALL receive_real_32bit(delu)
      CALL receive_real_32bit(delv)
      CALL receive_real_32bit(delw)
c***********************************************************************
c                                                       *
c                           partition 4                       *
c                                                       *
c***********************************************************************
c---------------------------- navigation module ----------------------c
      CALL navig(delphi, deltht, delpsi, delu, delv, delw, gr, t, sq,
     & sr, ti2m, at, delxd, delyd, delzd)
c  integrate performance velocity remaining using navigation output
      IF (t .LT. tst2on .OR. t .GE. tstg2) THEN
        mvrdot = 0.0
      ELSE
        mvrdot =  - sqrt(at(1)**2 + at(2)**2 + at(3)**2)
      ENDIF
      mvr = mvr + delt*mvrdot
      IF (mvr .LT. 0.0) mvr = 0.0
c************************************ ***********************************
c                                                       *
c                         end of partition 4                    *
c                                                       *
c***********************************************************************
c***********************************************************************
c                                                       *
c                           partition 5                       *
c                                                       *
c***********************************************************************
c--------------------on board guidance processing----------------------
---c
      IF (tstep .GE. tgpu) THEN
        tgpu = tgpu + dtbgu
        dt = t - tlgpu
        tlgpu = t
c  integrate guidance states from last pass through
        us(1) = us(1) + dt*usd(1)
        us(2) = us(2) + dt*usd(2)
        us(3) = us(3) + dt*usd(3)
        pg(1) = pg(1) + dt*pgd(1)
        pg(2) = pg(2) + dt*pgd(2)
```

```
        pg(3) = pg(3) + dt*pgd(3)
        vw(1) = vw(1) + dt*vwd(1)
        vw(2) = vw(2) + dt*vwd(2)
        vw(3) = vw(3) + dt*vwd(3)
        dtmp1 = sqrt(pg(1)**2 + pg(2)**2 + pg(3)**2)
        pg(1) = pg(1)/dtmp1
        pg(2) = pg(2)/dtmp1
        pg(3) = pg(3)/dtmp1
c------------------------------ boost steering module ------------------c
        IF (t .LT. tstg2) THEN
          CALL bsteer(t, us, uvs, mvs, mvr, at, usd, ac)
c------------------------------ boost guidance module ------------------c
          CALL bguid(t, at, ac, ti2m, pg, vw, pgd, vwd, psier, thter,
     &      pm)
        ELSE
          CALL receive_real_32bit(attlm)
        ENDIF
        IF (t .GE. tstg2) THEN
          usd(1) = 0.0
          usd(2) = 0.0
          usd(3) = 0.0
          pgd(1) = 0.0
          pgd(2) = 0.0
          pgd(3) = 0.0
          vwd(1) = 0.0
          vwd(2) = 0.0
          vwd(3) = 0.0
        ENDIF
      ELSE
        CALL receive_real_32bit(attlm)
      ENDIF
      CALL send_real_32bit(psier)
      CALL send_real_32bit(thter)
      CALL send_real_32bit(sq)
      CALL send_real_32bit(sr)
c*********************************************************************
c                                                          *
c                          end of partition 5                          *
c                                                          *
c*********************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 50

      END
```

A.1.13 Boost2c1.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL pulseg(3)
      REAL cg(3)
      REAL pulsea(3)
      REAL delu, delv, delw
#include "include/constant.dat"
      DATA delu/0.0/, delv/0.0/, delw/0.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

c-------------------------------------------------------------------------c
c----------------------------- main execution loop -----------------------c
c-------------------------------------------------------------------------c
   10 CONTINUE
c--------------- receive parameters from partition #1 -------------------c
      CALL receive_real_32bit(cg(1))
      CALL receive_real_32bit(cg(2))
      CALL receive_real_32bit(cg(3))
      CALL receive_real_32bit(p)
      CALL receive_real_32bit(q)
      CALL receive_real_32bit(r)
      CALL receive_real_32bit(ud)
      CALL receive_real_32bit(vd)
      CALL receive_real_32bit(wd)
      CALL receive_real_32bit(pd)
      CALL receive_real_32bit(qd)
      CALL receive_real_32bit(rd)
c*************************************************************************
c                                                       *
c                              partition 3                    *
c                                                       *
c*************************************************************************
c----------------------------- inertial measurement update ------------c
      CALL gyro(p, q, r, t, pulseg)
c----------------------------- inertial measurement update ------------c
      CALL accel(ud, vd, wd, p, q, r, pd, qd, rd, cg, t, pulsea)
c----------------------------- imu processor module -------------------c
      CALL imupro(pulsea, pulseg, delphi, deltht, delpsi, delu, delv,
     & delw)
      CALL send_real_32bit(delphi)
      CALL send_real_32bit(deltht)
      CALL send_real_32bit(delpsi)
      CALL send_real_32bit(delu)
      CALL send_real_32bit(delv)
      CALL send_real_32bit(delw)
c*************************************************************************
c                                                       *
c                           end of partition 3                      *
c                                                       *
c*************************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.14 Boost2c2.for

```
      PROGRAM main
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION vmir(3), rmir(3)
      DOUBLE PRECISION xyze(3)
      DOUBLE PRECISION xyzed(3)
      DOUBLE PRECISION cei(9), latlp, longlp
      REAL at(3), delxd, delyd, delzd, mvs, uvs(3)
      REAL t, tst2on, dtcvu, delt, tpico, tstg2
      REAL mvr, mvrdot, vtt(3), vpl
      REAL tcorv, vg(3), epsl
#include "include/constant.dat"
      DATA tst2on/22.995/
      DATA dtcvu/50.0/, epsl/2.5e-04/
      DATA dtr/0.017453292519943296/
      DATA latlp/0.0/, longlp/0.0/, vpl/13770.0/

c initialize time
      tstep = 0.0
      t = tstep * delt

      DO 10 i = 1, 3
        xyze(i) = 0.0
        xyzed(i) = 0.0
        vtt(i) = 0.0
        vg(i) = 0.0
   10 CONTINUE
      xyze(1) = 20898908.0
      vg(1) = 5000.0
      vg(3) = 9350.0
      mvrdot = 0.0
c-------------------------------------------------------------------------c
c----------------------------- missile state initialization module ----c
c-------------------------------------------------------------------------c
c   initialize states and state derivatives
      CALL mmk(-90.0*dtr, 1, latlp*dtr, 2, longlp*dtr, 3, cei)
      CALL vecrot(xyzed, cei, vmir)
      CALL vecrot(xyze, cei, rmir)
      CALL incorv(vg, rmir, vmir, mvs, uvs)
      mvr = vpl
      tcorv = 0.0
      istart = 0
c-------------------------------------------------------------------------c
c------------------------------ main execution loop --------------------c
c-------------------------------------------------------------------------c
   20 CONTINUE
c*************************************************************************
c                                                        *
c                             partition 4                      *
c                                                        *
c*************************************************************************
      CALL send_real_32bit(mvs)
      CALL send_real_32bit(uvs(1))
      CALL send_real_32bit(uvs(2))
      CALL send_real_32bit(uvs(3))
      CALL receive_real_32bit(at(1))
      CALL receive_real_32bit(at(2))
      CALL receive_real_32bit(at(3))
      CALL receive_real_32bit(delxd)
      CALL receive_real_32bit(delyd)
      CALL receive_real_32bit(delzd)
      IF (istart .EQ. 0) THEN
```

```
        istart = 1
      ELSE
        rmir(1) = rmir(1) + (vmir(1) + 0.5d0*delxd)*delt
        rmir(2) = rmir(2) + (vmir(2) + 0.5d0*delyd)*delt
        rmir(3) = rmir(3) + (vmir(3) + 0.5d0*delzd)*delt
        vmir(1) = vmir(1) + delxd
        vmir(2) = vmir(2) + delyd
        vmir(3) = vmir(3) + delzd
c  integrate performance velocity remaining using navigation output
        IF (t - delt .LT. tst2on .OR. t - delt .GE. tstg2) THEN
          mvrdot = 0.0
        ELSE
          mvrdot =  - sqrt(at(1)**2 + at(2)**2 + at(3)**2)
        ENDIF
        mvr = mvr + delt*mvrdot
        IF (mvr .LT. 0.0) mvr = 0.0
c  integrate gravity compensated acceleration
        vtt(1) = vtt(1) + delt*at(1)
        vtt(2) = vtt(2) + delt*at(2)
        vtt(3) = vtt(3) + delt*at(3)
      ENDIF
c------------------------------ correlated velocity module -------------c
      IF (tstep .GE. tcorv) THEN
        CALL corvel(mvr, t, vtt, rmir, vmir, vg, mvs, uvs)
        tcorv = tcorv + dtcvu
      ENDIF
c**************************************************************************
c                                                                        *
c                          end of partition 4                                *
c                                                                        *
c**************************************************************************

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 20

      END
```

A.1.15 Cg123.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL masst1(20), cgx(20), cgy(20), cgz(20)
      REAL mass, cg(3)
#include "include/constant.dat"
#include "include/masst1.dat"
#include "include/cgx.dat"
#include "include/cgy.dat"
#include "include/cgz.dat"
      DATA cg(1)/0.0/, cg(2)/0.0/, cg(3)/0.0/
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL send_real_32bit(cg(1))
      CALL send_real_32bit(cg(2))
      CALL send_real_32bit(cg(3))
      CALL receive_real_32bit(mass)

      CALL table(masst1, cgx, mass, cg(1), 20, itable)
      CALL table(masst1, cgy, mass, cg(2), 20, itable)
      CALL table(masst1, cgz, mass, cg(3), 20, itable)

      CALL send_real_32bit(cg(1))
      CALL send_real_32bit(cg(2))
      CALL send_real_32bit(cg(3))

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.16 Cne.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL cna1e(205), cna2e(205)
      REAL estmch, cne
#include "include/constant.dat"
#include "include/cna1e.dat"
#include "include/cna2e.dat"
      DATA icnm1e/0/, icna1e/0/
      DATA icnm2e/0/, icna2e/0/
      DATA tapu/0.0/, dtapu/5.0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(estmch)

      IF (tstep .GE. tapu) THEN
        tapu = tapu + dtapu
        IF (t .LT. tstg2) THEN
          IF (t .LT. tstg1) THEN
            CALL tlu2ei(estmch, 4.0e0, cna1e, icnm1e, icna1e, cne)
          ELSE
            CALL tlu2ei(estmch, 4.0e0, cna2e, icnm2e, icna2e, cne)
          ENDIF
        ENDIF
      ENDIF

      CALL send_real_32bit(cne)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.17 Inerxyz.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL masst1(20), inerxx(20), ineryy(20), inerzz(20)
      REAL ixx, iyy, izz
      REAL mass
#include "include/constant.dat"
#include "include/masst1.dat"
#include "include/inerxx.dat"
#include "include/ineryy.dat"
#include "include/inerzz.dat"
      DATA ixx/0.0/, iyy/0.0/, izz/0.0/
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL send_real_32bit(iyy)
      CALL receive_real_32bit(mass)

      CALL table(masst1, inerxx, mass, ixx, 20, itable)
      CALL table(masst1, ineryy, mass, iyy, 20, itable)
      CALL table(masst1, inerzz, mass, izz, 20, itable)

      CALL send_real_32bit(ixx)
      CALL send_real_32bit(iyy)
      CALL send_real_32bit(izz)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.18 Press.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL altt(59), presst(59)
#include "include/constant.dat"
#include "include/altt.dat"
#include "include/presst.dat"
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt

      press = 0.0

   10 CONTINUE
      CALL send_real_32bit(press)
      CALL receive_real_32bit(alt)

      CALL table(altt, presst, alt, press, 59, itable)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.19 Print.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
c     INCLUDE ':pfp:include/target.for'
      REAL garbage
      REAL t
      REAL alt, x, y, z
            REAL velocity, vrwmx, vrwmy, vrwmz
            REAL phi, tht, psi
#include "include/constant.dat"
      DATA tprt/0.0/, dtprt/100.0/

      CALL receive_real_32bit(garbage)

c initialize time
      tstep = 0.0
      t = tstep * delt

c-------------------------------------------------------------------------c
c----------------------------- main execution loop -----------------------c
c-------------------------------------------------------------------------c
   10 CONTINUE
c--------------- receive parameters from partition #1 --------------------c
            CALL receive_real_32bit(vrwmx)
            CALL receive_real_32bit(vrwmy)
            CALL receive_real_32bit(vrwmz)
            CALL receive_real_32bit(phi)
            CALL receive_real_32bit(tht)
            CALL receive_real_32bit(psi)
            CALL receive_real_32bit(x)
            CALL receive_real_32bit(y)
            CALL receive_real_32bit(z)
      CALL receive_real_32bit(alt)
      IF (tstep .GE. tprt) THEN
                  velocity = sqrt(vrwmx * vrwmx + vrwmy * vrwmy + vrwmz
* vrwmz)
                  CALL send_host_real(t)
                  CALL send_host_real(alt)
                  CALL send_host_real(x)
                  CALL send_host_real(y)
                  CALL send_host_real(z)
                  CALL send_host_real(velocity)
                  CALL send_host_real(vrwmx)
                  CALL send_host_real(vrwmy)
                  CALL send_host_real(vrwmz)
                  CALL send_host_real(phi)
                  CALL send_host_real(tht)
                  CALL send_host_real(psi)
         tprt = tprt + dtprt
      ENDIF

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10
      END
```

A.1.20 Rho.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL altt(59), rhot(59)
      REAL alt, rho
#include "include/constant.dat"
#include "include/constant.dat"
#include "include/altt.dat"
#include "include/rhot.dat"
      DATA itable/0/
      DATA slglbm/32.174048/

c initialize time
      tstep = 0.0
      t = tstep * delt

      rho = 0.0

   10 CONTINUE
      CALL receive_real_32bit(alt)

      CALL table(altt, rhot, alt, rho, 59, itable)
      rho = rho*1.0e-6/slglbm
      rhod2 = rho/2.0

      CALL send_real_32bit(rhod2)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.21 Shear.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL altt(59), sheart(59)
      REAL alt, shear
#include "include/constant.dat"
#include "include/altt.dat"
#include "include/sheart.dat"
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt

      shear = 0.0

   10 CONTINUE
      CALL receive_real_32bit(alt)

      CALL table(altt, sheart, alt, shear, 59, itable)

      CALL send_real_32bit(shear)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.22 Target.for

```
      PROGRAM main
c-----------------------------------------------------------------------
c      subroutine :          target(t,rtic,vtic)
c      function :            computes the rotational and translational
c                            target states
c      inputs :              t
c      outputs :             rtic,vtic
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION grt(3), rtic(3), vtic(3)
      DOUBLE PRECISION urtic(3), mrtic, mgrt
      INTEGER first1
#include "include/constant.dat"
      DATA first1/1/, gmu/1.4052477e16/
      DATA rtic/22462673.6, 0.0, 3781781.71/
      DATA vtic/-6858.46, 0.0, -18411.68/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit( alt )

      CALL magt(rtic, mrtic, urtic)
      mgrt = gmu/mrtic**2
      CALL mvbys(-mgrt, urtic, grt)
c  integrate target acceleration and velocity
      IF (first1 .EQ. 1) THEN
        first1 = 0
        tl1 = t
      ELSE
        tdelt = t - tl1
        tl1 = t
        DO 20 i = 1, 3
          rtic(i) = rtic(i) + vtic(i)*tdelt + 0.5d0*grt(i)*tdelt*tdelt
          vtic(i) = vtic(i) + grt(i)*tdelt
   20   CONTINUE
      ENDIF

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.23 Vsnd.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL altt(59), vsndt(59)
      REAL alt, vsnd
#include "include/constant.dat"
#include "include/altt.dat"
#include "include/vsndt.dat"
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(alt)

      CALL table(altt, vsndt, alt, vsnd, 59, itable)

      CALL send_real_32bit(vsnd)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.24 Vwind.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL altt(59), vwindt(59)
      REAL alt, vwind
#include "include/constant.dat"
#include "include/altt.dat"
#include "include/vwindt.dat"
      DATA itable/0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(alt)

      CALL table(altt, vwindt, alt, vwind, 59, itable)

      CALL send_real_32bit(vwind)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.25 Windir.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL altt(59), windirt(59)
      REAL alt, windir
#include "include/constant.dat"
#include "include/altt.dat"
#include "include/windirt.dat"
      DATA itable/0/
      DATA dtr/0.017453292519943296/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(alt)

      CALL table(altt, windirt, alt, windir, 59, itable)
      swdir = sin(windir*dtr)
      cwdir = cos(windir*dtr)

      CALL send_real_32bit(swdir)
      CALL send_real_32bit(cwdir)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.1.26 Xcpe.for

```
      PROGRAM main
      IMPLICIT REAL(a-h, o-z)
      REAL xcpl1e(205), xcpl2e(205)
      REAL estmch, alfate, xcpe
#include "include/constant.dat"
#include "include/xcpl1e.dat"
#include "include/xcpl2e.dat"
      DATA icpm1e/0/, icpa1e/0/
      DATA icpm2e/0/, icpa2e/0/
      DATA tapu/0.0/, dtapu/5.0/

c initialize time
      tstep = 0.0
      t = tstep * delt


   10 CONTINUE
      CALL receive_real_32bit(estmch)
      CALL receive_real_32bit(alfate)

      IF (tstep .GE. tapu) THEN
        tapu = tapu + dtapu
        IF (t .LT. tstg2) THEN
          IF (t .LT. tstg1) THEN
            CALL tlu2ei(estmch, alfate, xcpl1e, icpm1e, icpa1e, xcpe)
          ELSE
            CALL tlu2ei(estmch, alfate, xcpl2e, icpm2e, icpa2e, xcpe)
          ENDIF
        ENDIF
      ENDIF

      CALL send_real_32bit(xcpe)

c increment time
      tstep = tstep + 1.0
      t = tstep * delt

      IF (t .LT. tfinal) GOTO 10

      END
```

A.2 Utilities (FORTRAN)

A.2.1 Accel.for

```
      SUBROUTINE accel(ud, vd, wd, p, q, r, pd, qd, rd, cg, t, pulsea)
c-----------------------------------------------------------------------
c     function :            accelerometer model computes sensed delta
c                           velocity counts.  includes rotational
c                           effects,axis misalignment and nonorthogo-
c                           nality errors, scale factor errors, random
c                           and constant drift and quantization.
c     inputs :              ud,vd,wd,p,q,r,pd,qd,rd,cg,t
c     both :                pulsea
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL sfwia(3), sf1a(3), sf2a(3), qfraca(3), dca(3)
      REAL sfea(3), wdra(3), dvel(3), pulsea(3), pqr(3), pqrd(3)
      REAL limu(3), dum1(3), dum2(3), dum3(3), ximu(3)
      REAL cg(3)
      INTEGER gyseed
      DATA limu/-2.0, 2*0.0/, imuoff/0/, gyseed/123/, sppa/0.0/
      DATA qfraca/3*0.0/, sf1a, sf2a/6*0.0/, drmena/0.0/, dca/3*0.0/
      DATA thxya, thxza, thyxa, thyza, thzxa, thzya/6*0.0/
      DATA phia, thta, psia/3*0.0/, drsiga/0.0/
#include "../include/constant.dat"
c  sensor acceleration due to package offset from the cg
      IF (imuoff .EQ. 0) THEN
        udr = ud
        vdr = vd
        wdr = wd
      ELSE
        CALL spvecsub(cg, limu, ximu)
        CALL vmk(pd, qd, rd, pqrd)
        CALL vmk(p, q, r, pqr)
        CALL spcrosst(pqrd, ximu, dum1)
        CALL spcrosst(pqr, ximu, dum2)
        CALL spcrosst(pqr, dum2, dum3)
        udr = ud + dum1(1) + dum3(1)
        vdr = vd + dum1(2) + dum3(2)
        wdr = wd + dum1(3) + dum3(3)
      ENDIF
c  accelerometer axis misalignment effects
      udm = udr + vdr*psia - wdr*thta
      vdm =  - udr*psia + vdr + wdr*phia
      wdm = udr*thta - vdr*phia + wdr
c  accelerometer axis nonorthogonality effects
      udn = udm + vdm*thxza - wdm*thxya
      vdn =  - udm*thyza + vdm + wdm*thyxa
      wdn = udm*thzya - vdm*thzxa + wdm
c  compute scale factor errors
      sfwia(1) = sf1a(1)*udn + sf2a(1)*udn**2
      sfwia(2) = sf1a(2)*vdn + sf2a(2)*vdn**2
      sfwia(3) = sf1a(3)*wdn + sf2a(3)*wdn**2
      sfea(1) = udn + sfwia(1)
      sfea(2) = vdn + sfwia(2)
      sfea(3) = wdn + sfwia(3)
      IF (t .GT. 0.0) THEN
        DO 10 i = 1, 3
c  make a gaussian draw for random drift and add to constant drift
          CALL norm1(drsiga, drmena, gyseed, dra)
          wdra(i) = dra + dca(i)
c  compute delta velocity
```

```
            dvel(i) = delt*(sfea(i) + wdra(i))
            IF (sppa .GT. 0.0) THEN
c  unquantized output in counts
              qfraca(i) = qfraca(i) - pulsea(i) + dvel(i)/sppa
c  quantized output in counts
              pulsea(i) = aint(qfraca(i))
            ELSE
              pulsea(i) = dvel(i)
            ENDIF
   10     CONTINUE
        ELSE
c  initialize quantization output values to zero on first pass
          DO 20 i = 1, 3
            qfraca(i) = 0.0
            pulsea(i) = 0.0
   20     CONTINUE
        ENDIF
        RETURN
        END
```

A.2.2 Bguid.for

```
      SUBROUTINE bguid(t, at, ac, ti2m, pg, vw, pgd, vwd, psier, thter,
     & pm)
c-----------------------------------------------------------------------
c     function :            to calculate the error between the commanded
c                           pointing vector and the actual direction the
c                           missile is flying during boost
c     inputs :              t,at,ac,ti2m,pg
c     outputs :             pgd,vwd,psier,thter,pm
c     both :                vw
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL vwd(3), vw(3), wc(3), pgd(3), pg(3), at(3), ac
     & (3)
      REAL ti2m(9), ka, ka1, ka2, ka3, ka4, ka5
      REAL kv, kv1, kv2, kv3, kv4, kv5
      REAL pm(3), psier, thter, attlm
      DATA ka1/.0015/, ka2/.0015/, ka3/.0013/, ka4/.011/, ka5/.0045/
      DATA kv1/.00013/, kv2/.00013/, kv3/0.0/, kv4/0.0/, kv5/0.0/
      DATA tc/0.75/, ts/14.5/, t5/46.0/, t2s/58.2/, tcd
     & /60.2/
      DATA iminsf/0/, vwlim/10.0/, wlim/0.3/
#include "../include/constant.dat"
      IF (t .LE. ts) THEN
        ka = ka1
        kv = kv1
      ELSEIF (t .LE. tstg1) THEN
        ka = ka2
        kv = kv2
      ELSEIF (t .LE. t5) THEN
        ka = ka3
        kv = kv3
      ELSEIF (t .LE. t2s) THEN
        ka = ka4
        kv = kv4
      ELSE
        ka = ka5
        kv = kv5
      ENDIF
c  launch steering mode
      IF (t .LE. tc) THEN
        CALL spcrosst(at, ac, vwd)
        wc(1) = 0.0
        wc(2) = 0.0
        wc(3) = 0.0
        pgd(1) = 0.0
        pgd(2) = 0.0
        pgd(3) = 0.0
c  minimum impulse steering mode
      ELSEIF (t .LE. t5) THEN
        IF (iminsf .EQ. 0) THEN
          CALL xferb(ac, 3, pg)
          iminsf = 1
        ENDIF
        CALL spcrosst(at, ac, vwd)
        wc(1) = ka*vwd(1) + kv*vw(1)
        wc(2) = ka*vwd(2) + kv*vw(2)
        wc(3) = ka*vwd(3) + kv*vw(3)
        wcmax = amax1(abs(wc(1)), abs(wc(2)), abs(wc(3)))
        IF (wcmax .GT. wlim) THEN
          scale = wlim/wcmax
          CALL spmvbys(scale, wc, wc)
```

```
        ENDIF
        CALL spcrosst(wc, pg, pgd)
c  generalized energy management steering mode
      ELSEIF (t .LE. tcd) THEN
        CALL spcrosst(at, ac, vwd)
        vwmax = amax1(abs(vw(1)), abs(vw(2)), abs(vw(3)))
        IF (vwmax .GT. vwlim) THEN
          scale = vwlim/vwmax
          CALL spmvbys(scale, vw, vw)
        ENDIF
        wc(1) = ka*vwd(1) + kv*vw(1)
        wc(2) = ka*vwd(2) + kv*vw(2)
        wc(3) = ka*vwd(3) + kv*vw(3)
        wcmax = amax1(abs(wc(1)), abs(wc(2)), abs(wc(3)))
        IF (wcmax .GT. wlim) THEN
          scale = wlim/wcmax
          CALL spmvbys(scale, wc, wc)
        ENDIF
        CALL spcrosst(wc, pg, pgd)
c  countdown steering mode
      ELSE
        CALL spcrosst(at, ac, vwd)
        vwmax = amax1(abs(vw(1)), abs(vw(2)), abs(vw(3)))
        IF (vwmax .GT. vwlim) THEN
          scale = vwlim/vwmax
          CALL spmvbys(scale, vw, vw)
        ENDIF
        wc(1) = 0.0
        wc(2) = 0.0
        wc(3) = 0.0
        pgd(1) = 0.0
        pgd(2) = 0.0
        pgd(3) = 0.0
      ENDIF
      call spvecrot(pg,ti2m,pm)
      psier = pm(2)
      thter =  - pm(3)
      CALL receive_real_32bit(attlm)
      toterr = sqrt(psier**2 + thter**2)
      IF (toterr .GT. attlm) THEN
        psier = psier*attlm/toterr
        thter = thter*attlm/toterr
      ENDIF
      RETURN
      END
```

A.2.3 Bsteer.for

```
      SUBROUTINE bsteer(t, us, uvs, mvs, mvr, at, usd, ac)
c-----------------------------------------------------------------------
c     function :            calculates the steering commands for the
boost
c                           phase of flight
c     inputs :              t,uvs,mvs,mvr,at
c     outputs :             usd,ac
c     both :                us
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL usf(3), uvs(3), dbar(3), usd(3), at(3)
      REAL us(3), ac(3), bbar(3), bigac(3), bigb(3), mvs,
     & mvr
      REAL ks1, kb, mbigac, mbigb
      DATA ks1/0.1/, kb/2.4495/, dbar/0.0, -1.0, 0.0/
      DATA tc/0.75/, t5/46.0/, tcd/60.2/, usfd/-35.0/, psi0/0.0/
      DATA dtr/0.017453292519943296/
c  launch steering logic
      IF (t .LE. tc) THEN
        usd(1) = 0.0
        usd(2) = 0.0
        usd(3) = 0.0
        CALL xferb(us, 3, ac)
        usf(1) = cos(psi0)*cos(usfd*dtr)
        usf(2) = sin(psi0)*cos(usfd*dtr)
        usf(3) =  - sin(usfd*dtr)
c  minimum impulse steering (mins) logic
      ELSEIF (t .LE. t5) THEN
        CALL dott(us, usf, usdot)
        usd(1) = ks1*(usf(1) - usdot*us(1))
        usd(2) = ks1*(usf(2) - usdot*us(2))
        usd(3) = ks1*(usf(3) - usdot*us(3))
        CALL xferb(us, 3, ac)
c  general energy management (gems) steering logic
      ELSEIF (t .LE. tcd) THEN
        usd(1) = 0.0
        usd(2) = 0.0
        usd(3) = 0.0
        CALL xferb(uvs, 3, us)
        CALL spcrosst(dbar, us, bigb)
        CALL spmagt(bigb, mbigb, bbar)
        IF (mvr .NE. 0.0) vratio = mvs/mvr
        IF (mvs .LE. mvr) THEN
          wastan = kb*(1.0 - vratio)**0.5
        ELSE
          wastan = 0.0
        ENDIF
        sinwan = vratio*wastan
        coswan = (1.0 - (wastan**2/2.0))
        bigac(1) = us(1)*coswan - bbar(1)*sinwan
        bigac(2) = us(2)*coswan - bbar(2)*sinwan
        bigac(3) = us(3)*coswan - bbar(3)*sinwan
        CALL spmagt(bigac, mbigac, ac)
c  countdown steering logic
      ELSE
        CALL xferb(us, 3, ac)
      ENDIF
      RETURN
      END
```

A.2.4 Bthrst.for

```
      SUBROUTINE bthrst(t, cg, press, dlp, dly, fxt, fyt, fzt, mxt,
     & myt, mzt, mdott)
c-----------------------------------------------------------------------
c     function :            computes missile thrust vector and moments
c                           due to first and second stage boosters
c     inputs :              t,cg,press,dlp,dly
c     outputs :             fxt,fyt,fzt,mxt,myt,mzt,mdott
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL timth1(26), thrth1(26)
      REAL timth2(29), thrth2(29)
      REAL thrma(9)
      REAL mxt, myt, mzt, mdott
      REAL cg(3), press
#include "../include/constant.dat"
#include "../include/timth1.dat"
#include "../include/thrth1.dat"
#include "../include/timth2.dat"
#include "../include/thrth2.dat"
      DATA tign/0.01/, tst2on/22.995/
      DATA bisp1/273.7/, bisp2/282.3/, wins1/8.19/, wins2/6.93/
      DATA xnoz/-12.5583/, xnoz2/-7.39167/, ib, ith1, ith2/3*0/
      DATA wprop1/710.28/, wprop2/479.61/, aexit/.305/, aexit2/0.99/
      DATA thrma/1.0, 3*0.0, 1.0, 3*0.0, 1.0/, slglbm/32.174048/
c  post boost
      IF (t .GE. tstg2) THEN
        fxt = 0.0e0
        fyt = 0.0e0
        fzt = 0.0e0
        mxt = 0.0e0
        myt = 0.0e0
        mzt = 0.0e0
        mdott = 0.0e0
      ELSE
        IF (t .GE. tstg1) THEN
          t0 = t - tst2on
          CALL table(timth2, thrth2, t0, thrv, 29, ith2)
c  first stage boost
        ELSE
          t0 = t - tign
          CALL table(timth1, thrth1, t0, thrv, 26, ith1)
        ENDIF
        IF (ib .EQ. 0) THEN
          eisp = bisp1*wprop1/(wprop1 + wins1)
          ib = 1
        ENDIF
        IF (abs(t - tstg1) .LE. dteps) THEN
          aexit = aexit2
          xnoz = xnoz2
          eisp = bisp2*wprop2/(wprop2 + wins2)
        ENDIF
        thr = max1(0.0e0, thrv - aexit*press)
        fx = thr*cos(dlp)*cos(dly)
        fy =  - thr*sin(dly)
        fz = thr*sin(dlp)*cos(dly)
        fxt = fx*thrma(1) + fy*thrma(4) + fz*thrma(7)
        fyt = fx*thrma(2) + fy*thrma(5) + fz*thrma(8)
        fzt = fx*thrma(3) + fy*thrma(6) + fz*thrma(9)
        mxt = fyt*cg(3) - fzt*cg(2)
        myt =  - fxt*cg(3) + fzt*(cg(1) - xnoz)
        mzt = fxt*cg(2) - fyt*(cg(1) - xnoz)
```

```
  mdott = thrv/(eisp*slglbm)
ENDIF
RETURN
END
```

A.2.5 Bxi2fv.for

```
      SUBROUTINE bxi2fv(fvm, b, fv)
C-----------------------------------------------------------------------
C     function :            compute quaternion (fv) attitude parameters
C                           from a body to inertial transformation
C                           matrix (b) and set the square of magnitude
C                           of quaternion to (fvm)
C     inputs :              fvm,b
C     outputs :             fv
C-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      DIMENSION b(9), fv(4)
      EQUIVALENCE (t3, q), (b1, aa)
      DATA f4, f2, p25, p0001/4., 2., 0.25, 0.0001/
      DATA f1, f0/1., 0./
      t3 = p25
      a1 = b(6) - b(8)
      a2 = b(7) - b(3)
      a3 = b(2) - b(4)
      tra = b(1) + b(5) + b(9) + f1
      IF (tra .LT. p0001) THEN
        iflag = 0
        tra = f2 - tra
        b1 = t3*(b(1) + b(1) + tra)
        IF (b1 .LT. f0) b1 = f0
        fv(1) = sqrt(b1)
        IF (fv(1) .NE. f0) iflag = 1
        b1 = t3*(b(5) + b(5) + tra)
        IF (b1 .LT. f0) b1 = f0
        fv(2) = sqrt(b1)
        IF (iflag .EQ. 1) fv(2) = sign(fv(2), b(2) + b(4))
        b1 = t3*(b(9) + b(9) + tra)
        IF (b1 .LT. f0) b1 = f0
        fv(3) = sqrt(b1)
        IF (iflag .EQ. 1) fv(3) = sign(fv(3), b(3) + b(7))
        IF (iflag .NE. 1) THEN
          IF (fv(2) .NE. f0) fv(3) = sign(fv(3), b(6) + b(8))
        ENDIF
        aa = f0
        fv(4) = f0
        q = f0
        IF (fv(1) .NE. f0) THEN
          q = f4
          aa = aa + a1/fv(1)
        ENDIF
        IF (fv(2) .NE. f0) THEN
          q = q + f4
          aa = aa + a2/fv(2)
        ENDIF
        IF (fv(3) .NE. f0) THEN
          q = q + f4
          aa = aa + a3/fv(3)
        ENDIF
        IF (q .NE. f0) fv(4) = aa/q
      ELSE
        fv(4) = sqrt(t3*tra)
        t3 = t3/fv(4)
        fv(1) = t3*a1
        fv(2) = t3*a2
        fv(3) = t3*a3
      ENDIF
      q = sqrt(fv(1)**2 + fv(2)**2 + fv(3)**2 + fv(4)**2)
```

```
IF (q .NE. f0) THEN
  q = fvm/q
  fv(1) = q*fv(1)
  fv(2) = q*fv(2)
  fv(3) = q*fv(3)
  fv(4) = q*fv(4)
ENDIF
RETURN
END
```

A.2.6 Corvel.for

```
      SUBROUTINE corvel(mvr, t, vtt, rmir, vmir, vg, mvs, uvs)
c-----------------------------------------------------------------------
c     function :            calculates the correlated velocity
c     inputs :              mvr,t,vtt,rmir,vmir
c     outputs :             mvs,uvs
c     both :                vg
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION rb(3), vc(3)
      DOUBLE PRECISION urb(3), urt(3), tmpv(3), utmpv(3), uthp(3)
      DOUBLE PRECISION vphi(3), rtpred(3)
      DOUBLE PRECISION vce(3), vse(3)
      DOUBLE PRECISION dlv(3), rmir(3), vmir(3)
      DOUBLE PRECISION mrb, mrt, mtmpv, mvce
      DOUBLE PRECISION mvse, mdvt
      REAL vge(3), vttp(3)
      REAL vd0(3), mvs, uvs(3), vs(3), t, mvr, tstg2, vg(3), vtt(3)
      DATA rtpred/21227680.0, 0.0, 1168230.0/, ttf/140.0/
      DATA f1/450.0/, f2/-0.5/, vd0/3*0.0/, dlv/3*0.0/, vttp/3*0.0/
      DATA gmu/1.4052477e16/
#include "../include/constant.dat"
c  estimate velocity to be gained (vge) , correlated velocity (vce) ,
c     and steering velocity (vse)
      DO 10 i = 1, 3
        dlv(i) = vtt(i) - vttp(i)
        vge(i) = vg(i) - dlv(i)
        vce(i) = vge(i) + vmir(i)
        vse(i) = vge(i) - vd0(i)
        vttp(i) = vtt(i)
   10 CONTINUE
      mvse = dsqrt(vse(1)**2 + vse(2)**2 + vse(3)**2)
      mdvt = dsqrt(dlv(1)**2 + dlv(2)**2 + dlv(3)**2)
      IF (mvse .GT. mvr) THEN
        scale3 = mvr/mvse
      ELSE
        scale3 = 1.0
      ENDIF
      scalar = f2*mvr*scale3/(f1 + mdvt)
      IF (t .GE. tstg2) THEN
        rb(1) = rmir(1)
        rb(2) = rmir(2)
        rb(3) = rmir(3)
      ELSE
        rb(1) = rmir(1) + scalar*vse(1)
        rb(2) = rmir(2) + scalar*vse(2)
        rb(3) = rmir(3) + scalar*vse(3)
      ENDIF
      CALL magt(rb, mrb, urb)
      CALL magt(rtpred, mrt, urt)
      CALL crosst(urb, urt, tmpv)
      CALL magt(tmpv, mtmpv, utmpv)
      CALL crosst(utmpv, urb, uthp)
      vhc = vce(1)*uthp(1) + vce(2)*uthp(2) + vce(3)*uthp(3)
      vcr = vce(1)*urb(1) + vce(2)*urb(2) + vce(3)*urb(3)
      CALL crosst(urb, urt, vphi)
      sinphi = dsqrt(vphi(1)**2 + vphi(2)**2 + vphi(3)**2)
      cosphi = urb(1)*urt(1) + urb(2)*urt(2) + urb(3)*urt(3)
      mvce = dsqrt(vce(1)**2 + vce(2)**2 + vce(3)**2)
      w = vhc/mrb
      el = mrb*vhc**2/gmu
      ar = mrb/mrt
```

```
      tp1 = mvce**2*mrb/gmu
      hhh = el*sinphi**2*(2.0 - tp1)
      sqrhhh = dsqrt(hhh)
      t1 = el*sinphi/(hhh*w)
      t2a = (1.0 - el)/ar + 1.0 - ar*el
      t2b = (2.0*el - 1.0 - 1.0/ar)*cosphi
      t2 = t2a + t2b
      t3 = 2.0*el**2*sinphi**3/(w*hhh*sqrhhh)
      t4a = sqrhhh
      t4b = el + ar*el + cosphi - 1.0
      t4 = arctan(t4a, t4b)
      tffe = t1*t2 + t3*t4
      ttfe = t + tffe
      tff = ttf - t
      deltf = tff - tffe
      a = 2.0*(ar - cosphi)/sinphi + (vcr/vhc)
      b = a*vcr - vhc
      c = b*mrb/gmu
      d = c*el*sinphi**2
      e = d + hhh/vhc
      parhv = e*2.0
      part1v = (1.0/vhc - parhv/hhh)*t1
      part2v = (2.0*el/vhc)*(2.0*cosphi - (1.0 + ar**2)/ar)
      part3v = (1.0/vhc - parhv/(2.0*hhh))*3.0*t3
      subeq1 = (el + ar*el + cosphi - 1.0)*vhc*parhv
      subeq2 = 4.0*hhh*el*(1.0 + ar)
      subeq3 = (el + ar*el + cosphi - 1.0)**2 + hhh
      subeq4 = 2.0*sqrhhh*vhc
      part4v = (subeq1 - subeq2)/(subeq3*subeq4)
      ptffv = t1*part2v + t2*part1v + t3*part4v + t4*part3v
      vcopk = vhc + deltf/ptffv
      vcrpk = (vcopk/(el*sinphi))*(1.0 - ar*el - (1.0 - el)*cosphi)
      DO 20 j = 1, 3
        vc(j) = vcrpk*urb(j) + vcopk*uthp(j)
        vg(j) = vc(j) - vmir(j)
        vs(j) = vg(j) - vd0(j)
   20 CONTINUE
      CALL spmagt(vs, mvs, uvs)
      RETURN
      END
```

A.2.7 Crosst.for

```
      SUBROUTINE crosst(v1, v2, r)
c-------------------------------------------------------------------------
c     function :            takes the cross product of two vectors
c
c                           ( v1 x v2 ) = r
c     inputs :              v1, v2
c     outputs :             r
c-------------------------------------------------------------------------
      DOUBLE PRECISION v1(3), v2(3), r(3)
      r(1) = v1(2)*v2(3) - v1(3)*v2(2)
      r(2) = v1(3)*v2(1) - v1(1)*v2(3)
      r(3) = v1(1)*v2(2) - v1(2)*v2(1)
      RETURN
      END
```

A.2.8 Dott.for

```
      SUBROUTINE dott(v1, v2, r)
c-----------------------------------------------------------------------
c     function :            takes the dot product of two vectors
c     inputs :              v1,v2
c     outputs :             r
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      DIMENSION v1(3), v2(3)
      r1 = v1(1)*v2(1)
      r2 = v1(2)*v2(2)
      r3 = v1(3)*v2(3)
      r = r1 + r2 + r3
      RETURN
      END
```

A.2.9 Fracs.for

```
      SUBROUTINE fracs(t  dlpc, dlyc, sq, sr, mdltfr, malpha, pm, tmf,
     & thf, lenf)
c-----------------------------------------------------------------------
c     function :            models the hysteresis function employed
c                           to generate the thruster commands
c     inputs :              t,dlpc,dlyc,sq,sr,mdltfr,malpha,pm
c     both :                tmf,thf,lenf
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL vcmd(4), vcmdl(4), kdtoff
      REAL kdelta, tmodtb(4), kdeltb(4)
      REAL mdltfr, malpha, dtoff(4)
      REAL pm(3), tmf(10, 4), thf(10, 4)
      REAL tmfl(10, 4), thfl(10, 4)
      INTEGER vcod(4), vcodl(4), vlvcm5
      INTEGER lenf(4)
      DATA dtfru/0.005/, tmode2/23.01/, t5/46.0/, delon/0.045/
      DATA deloff/0.035/, realmn/1.e-20/, thjet/370./
      DATA tlagfr, trupfr, trdnfr/3*0.00125/, vlvcm5/0/, ikdel/0/
      DATA tmodtb/0.00, 22.4, 39.95, 100.0/, kdeltb/0.4, 0.88, 2.05,
     & 2.05/
      DATA kdtoff/1.2/, bdtoff/0.0/, vcmd/4*0.0/, vcod/4*0/, dtoff/4
     & *0.0/
      DO 10 i = 1, 4
        vcmdl(i) = vcmd(i)
        vcodl(i) = vcod(i)
   10 CONTINUE
      CALL table(tmodtb, kdeltb, t - tmode2, kdelta, 4, ikdel)
      delonp = kdelta*delon
      delofp = kdelta*deloff
      psense = sign(1.0e0, dlpc)
      ysense = sign(1.0e0, dlyc)
      IF (psense .LT. 0.0) THEN
        IF (vcmdl(4) .GT. 0.5e0) THEN
          IF (abs(dlpc) .LE. delofp) THEN
            vcmd(4) = 0.0
          ELSE
            vcmd(4) = 1.0
          ENDIF
        ELSEIF (abs(dlpc) .GE. delonp) THEN
          vcmd(4) = 1.0
        ELSE
          vcmd(4) = 0.0
        ENDIF
        vcmd(2) = 0.0
      ELSE
        IF (vcmdl(2) .GT. 0.5) THEN
          IF (dlpc .LE. delofp) THEN
            vcmd(2) = 0.0
          ELSE
            vcmd(2) = 1.0
          ENDIF
        ELSEIF (dlpc .GE. delonp) THEN
          vcmd(2) = 1.0
        ELSE
          vcmd(2) = 0.0
        ENDIF
        vcmd(4) = 0.0
      ENDIF
      IF (ysense .LT. 0.0) THEN
        IF (vcmdl(1) .GT. 0.5) THEN
```

```
        IF (abs(dlyc) .LE. delofp) THEN
          vcmd(1) = 0.0
        ELSE
          vcmd(1) = 1.0
        ENDIF
      ELSEIF (abs(dlyc) .GE. delonp) THEN
        vcmd(1) = 1.0
      ELSE
        vcmd(1) = 0.0
      ENDIF
      vcmd(3) = 0.0
    ELSE
      IF (vcmdl(3) .GT. 0.5) THEN
        IF (dlyc .LE. delofp) THEN
          vcmd(3) = 0.0
        ELSE
          vcmd(3) = 1.0
        ENDIF
      ELSEIF (dlyc .GE. delonp) THEN
        vcmd(3) = 1.0
      ELSE
        vcmd(3) = 0.0
      ENDIF
      vcmd(1) = 0.0
    ENDIF
    vlvcm5 = vlvcm5 + aint(vcmd(1) + vcmd(2) + vcmd(3) + vcmd(4))
    DO 30 i = 1, 4
      IF (t .GE. t5) THEN
        IF (i .EQ. 1) THEN
          dt1frc = abs(sr)/(mdltfr - sqrt(malpha)*abs(sr))
          dtoff(1) = kdtoff*2.0*abs(dt1frc) + bdtoff
          dtoff(1) = aint(dtoff(1)*10000.0)*0.0001
          dtoff(3) = dtoff(1)
        ELSEIF (i .EQ. 2) THEN
          dt1frc = abs(sq)/(mdltfr - sqrt(malpha)*abs(sq))
          dtoff(2) = kdtoff*2.0*abs(dt1frc) + bdtoff
          dtoff(2) = aint(dtoff(2)*10000.0)*0.0001
          dtoff(4) = dtoff(2)
        ENDIF
      ELSEIF (i .EQ. 1) THEN
        temp = mdltfr - malpha*abs(pm(2)) + realmn
        dt1frc =  - sr/temp
        dt2frc = sqrt(malpha)*( - pm(2))/temp
        dtoff(1) = kdtoff*(abs(dt1frc) + abs(dt2frc)) + bdtoff
        dtoff(1) = aint(dtoff(1)*10000.0)*0.0001
        dtoff(3) = dtoff(1)
      ELSEIF (i .EQ. 2) THEN
        temp = mdltfr - malpha*abs(pm(3)) + realmn
        dt1frc =  - sq/temp
        dt2frc = sqrt(malpha)*( - pm(3))/temp
        dtoff(2) = kdtoff*(abs(dt1frc) + abs(dt2frc)) + bdtoff
        dtoff(2) = aint(dtoff(2)*10000.0)*0.0001
        dtoff(4) = dtoff(2)
      ENDIF
      DO 20 j = 1, 10
        tmfl(j, i) = tmf(j, i)
        thfl(j, i) = thf(j, i)
        tmf(j, i) = 0.0
        thf(j, i) = 0.0
20    CONTINUE
      IF (vcmd(i) .GT. 0.5) THEN
        IF (vcmdl(i) .GT. 0.5) THEN
          IF (vcodl(i) .EQ. 3) THEN
            tmf(1, i) = t
```

```
  thf(1, i) = thjet
  iptr = 2
ELSEIF (vcodl(i) .EQ. 0) THEN
  tmf(1, i) = t
  thf(1, i) = 0.0
  tmf(2, i) = t + tlagfr
  thf(2, i) = 0.0
  tmf(3, i) = tmf(2, i) + trupfr
  thf(3, i) = thjet
  iptr = 4
ELSE
  tmf(1, i) = tmfl(lenf(i) - 2, i)
  thf(1, i) = thfl(lenf(i) - 2, i)
  tmf(2, i) = tmfl(lenf(i) - 1, i)
  thf(2, i) = thfl(lenf(i) - 1, i)
  tmf(3, i) = tmfl(lenf(i), i)
  thf(3, i) = thfl(lenf(i), i)
  IF (vcodl(i) .EQ. 2) THEN
    tt1 = t + tlagfr
    tt2 = tt1 + trupfr
    IF (tmf(3, i) .GT. tt1) THEN
      tmf(3, i) = (tmf(3, i) + tt1)/2.0
      thf(3, i) = (tmf(3, i) - tt1)*thjet/trdnfr
      tmf(4, i) = tt2
      thf(4, i) = thjet
      iptr = 5
    ELSE
      tmf(4, i) = tt1
      thf(4, i) = 0.0
      tmf(5, i) = tt2
      thf(5, i) = thjet
      iptr = 6
    ENDIF
  ELSE
    tmf(4, i) = t + tlagfr
    thf(4, i) = 0.0
    tmf(5, i) = tmf(4, i) + trupfr
    thf(5, i) = thjet
    iptr = 6
  ENDIF
ENDIF
IF (dtoff(i) .GE. dtfru) THEN
  tmf(iptr, i) = t + dtfru
  thf(iptr, i) = thjet
  lenf(i) = iptr
  vcod(i) = 3
ELSEIF ((dtoff(i) + tlagfr) .GE. dtfru) THEN
  tmf(iptr, i) = t + dtoff(i)
  thf(iptr, i) = thjet
  tmf(iptr + 1, i) = tmf(iptr, i) + tlagfr
  thf(iptr + 1, i) = thjet
  tmf(iptr + 2, i) = tmf(iptr + 1, i) + trdnfr
  thf(iptr + 2, i) = 0.0
  lenf(i) = iptr + 2
  vcod(i) = 2
ELSEIF ((dtoff(i) + tlagfr + trupfr) .GE. dtfru) THEN
  tmf(iptr, i) = t + dtoff(i) + tlagfr
  thf(iptr, i) = thjet
  tmf(iptr + 1, i) = tmf(iptr, i) + trdnfr
  thf(iptr + 1, i) = 0.0
  lenf(i) = iptr + 1
  vcod(i) = 1
ELSE
  tmf(iptr, i) = t + tlagfr + trupfr + tlagfr
```

```
        thf(iptr, i) = thjet
        tmf(iptr + 1, i) = tmf(iptr, i) + trdnfr
        thf(iptr + 1, i) = 0.0
        lenf(i) = iptr + 1
        vcod(i) = 0
      ENDIF
    ELSE
      tmf(1, i) = t
      thf(1, i) = 0.0
      tmf(2, i) = t + tlagfr
      thf(2, i) = 0.0
      tmf(3, i) = tmf(2, i) + trupfr
      thf(3, i) = thjet
      IF (dtoff(i) .GE. dtfru) THEN
        tmf(4, i) = t + dtfru
        thf(4, i) = thjet
        lenf(i) = 4
        vcod(i) = 3
      ELSEIF ((dtoff(i) + tlagfr) .GE. dtfru) THEN
        tmf(4, i) = t + dtoff(i) + tlagfr
        thf(4, i) = thjet
        tmf(5, i) = tmf(4, i) + trdnfr
        thf(5, i) = 0.0
        lenf(i) = 5
        vcod(i) = 2
      ELSEIF ((dtoff(i) + tlagfr + trupfr) .GT. dtfru) THEN
        tmf(4, i) = t + dtoff(i) + tlagfr
        thf(4, i) = thjet
        tmf(5, i) = tmf(4, i) + trdnfr
        thf(5, i) = 0.0
        lenf(i) = 5
        vcod(i) = 1
      ELSE
        tmf(4, i) = tmf(3, i) + tlagfr
        thf(4, i) = thjet
        tmf(5, i) = t + dtfru
        thf(5, i) = 0.0
        lenf(i) = 5
        vcod(i) = 0
      ENDIF
    ENDIF
  ELSEIF (vcmdl(i) .GT. 0.5) THEN
    IF (vcodl(i) .EQ. 3) THEN
      tmf(1, i) = t
      thf(1, i) = thjet
      tmf(2, i) = tmf(1, i) + tlagfr
      thf(2, i) = thjet
      tmf(3, i) = tmf(2, i) + trdnfr
      thf(3, i) = 0.0
      tmf(4, i) = t + dtfru
      thf(4, i) = 0.0
      lenf(i) = 4
    ELSEIF (vcodl(i) .EQ. 0) THEN
      tmf(1, i) = t
      thf(1, i) = 0.0
      tmf(2, i) = t + dtfru
      thf(2, i) = 0.0
      lenf(i) = 2
    ELSE
      tmf(1, i) = tmfl(lenf(i) - 2, i)
      thf(1, i) = thfl(lenf(i) - 2, i)
      tmf(2, i) = tmfl(lenf(i) - 1, i)
      thf(2, i) = thfl(lenf(i) - 1, i)
      tmf(3, i) = tmfl(lenf(i), i)
```

```
            thf(3, i) = thfl(lenf(i), i)
            tmf(4, i) = t + dtfru
            thf(4, i) = 0.0
            lenf(i) = 4
          ENDIF
          vcod(i) = 0
        ENDIF
30    CONTINUE
      RETURN
      END
```

A.2.10 Frcthr.for

```
      SUBROUTINE frcthr(t, cg, mach, qa, tmf, thf, lenf, frcx, frcy,
     & frcz, mrcx, mrcy, mrcz, mdotf)
c-----------------------------------------------------------------------
c     function :            computes forces and moments resulting from
c                           the forward reaction control thrusters
c     inputs :              t,cg,mach,qa,tmf,thf,lenf
c     outputs :             frcx,frcy,frcz,mrcx,mrcy,mrcz,mdotf
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL frcdir(3, 4), frcloc(3, 4), frcma(9, 4)
      REAL f0(3)
      REAL f(3), xmom(3), m(3)
      REAL mrcx, mrcy, mrcz
      REAL mchlim, kn
      REAL km, ld, mdotf
      REAL tmf(10, 4), thf(10, 4)
      REAL athrf(4), knfac
      REAL kmfac
      REAL mach, cg(3)
      INTEGER indx(4), lenf(4)
#include "../include/constant.dat"
#include "../include/frcdir.dat"
#include "../include/frcloc.dat"
#include "../include/frcma.dat"
      DATA indx/4*0/, thjet/370./, sjet/1.3273/
      DATA djet/1.3/, fisp/281.3/, knfac/1.0/, kmfac/0.0/, mchlim/4.0/
      DATA xjet/-2.71/
      DATA xnoz/-12.55837/, xnoz2/-7.39167/, slglbm/32.174048/
      IF (abs(t - tstg1) .LE. dteps) xnoz = xnoz2
      frcx = 0.0e0
      frcy = 0.0e0
      frcz = 0.0e0
      mrcx = 0.0e0
      mrcy = 0.0e0
      mrcz = 0.0e0
      mdotf = 0.0e0
      ld = (xjet - xnoz)/djet
      ct = thjet/(qa*sjet)
      IF (mach .LE. mchlim) THEN
        kn = 0.6118 + (0.1358*(1. - 0.485*sqrt(ld))/sqrt(ct)) + 0.0946
     &   *mach + 0.004317/ld
      ELSE
        kn = 1.0 + exp(1.1 - 0.2116*(log(ct) + 8.5)**1.4)
      ENDIF
      km = 0.5582 - 0.1884/sqrt(ct) - 1.9659/ld
      tref = t
      DO 20 i = 1, 4
        IF (tmf(1, i) .GT. 0.0e0) THEN
          CALL table(tmf(1, i), thf(1, i), tref, athrf(i), lenf(i),
     &     indx(i))
        ELSE
          athrf(i) = 0.0e0
          indx(i) = 0
        ENDIF
        athrf(i) = athrf(i)/thjet
        DO 10 j = 1, 3
          f0(j) = frcdir(j, i)*kn*knfac*thjet*athrf(i)
          xmom(j) = cg(j) - frcloc(j, i)
   10   CONTINUE
        f(1) = f0(1)*frcma(1, i) + f0(2)*frcma(4, i) + f0(3)*frcma(7,
     &   i)
```

```
      f(2) = f0(1)*frcma(2, i) + f0(2)*frcma(5, i) + f0(3)*frcma(8,
     &   i)
      f(3) = f0(1)*frcma(3, i) + f0(2)*frcma(6, i) + f0(3)*frcma(9,
     &   i)
      CALL spcrosst(f, xmom, m)
      frcx = frcx + f(1)
      frcy = frcy + f(2)
      frcz = frcz + f(3)
      mrcx = mrcx + m(1)
      mrcy = mrcy + m(2)
      mrcz = mrcz + m(3)
      IF (i .EQ. 1 .OR. i .EQ. 3) THEN
        mrcy = mrcy + frcdir(3, i)*thjet*km*kmfac*djet*athrf(i)
15    ELSE
        mrcz = mrcz - frcdir(2, i)*thjet*km*kmfac*djet*athrf(i)
      ENDIF
      mdotf = mdotf + thjet*athrf(i)/(fisp*slglbm)
20  CONTINUE
    RETURN
    END
```

A.2.11 Fv2bxi.for

```
      SUBROUTINE fv2bxi(fv, fvsq, b)
c-----------------------------------------------------------------------
c     function :            compute direction cosine matrix (b) from
c                           the quaternion attitude vector (fv) and
c                           compute the square (fvsq) of the magnitude
c                           of the quaternion (fv)
c     inputs :              fv
c     outputs :             fvsq,b
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      DIMENSION fv(4), b(9)
      DATA r1, r2/1.0, 2.0/
      f1 = fv(1)
      f2 = fv(2)
      f3 = fv(3)
      f4 = fv(4)
      f1s = f1*f1
      f2s = f2*f2
      f3s = f3*f3
      f4s = f4*f4
      tt = f1s + f2s + f3s + f4s
      IF (tt .GT. 0) THEN
        t1 = r2/tt
        t2 = f3*f4
        t3 = f1*f2
        b(2) = t1*(t3 + t2)
        b(4) = t1*(t3 - t2)
        t2 = f2*f4
        t3 = f1*f3
        b(7) = t1*(t3 + t2)
        b(3) = t1*(t3 - t2)
        t2 = f1*f4
        t3 = f2*f3
        b(6) = t1*(t3 + t2)
        b(8) = t1*(t3 - t2)
        t2 = t1*f4s - r1
        b(1) = t1*f1s + t2
        b(5) = t1*f2s + t2
        b(9) = t1*f3s + t2
      ENDIF
      fvsq = tt
      RETURN
      END
```

A.2.12 Fvdot.for

```
      SUBROUTINE fvdot(w, wd, f, fd)
c-----------------------------------------------------------------------
c     function :            compute the quaternion derivatives (fd)
c                           using body rates (w) and latent integral
c                           derivative (wd) and the quaternion (f)
c     inputs :              w,wd,f
c     outputs :             fd
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      DIMENSION w(3), f(4), fd(4)
      w1 = w(1)
      w2 = w(2)
      w3 = w(3)
      w4 = wd
      f1 = f(1)
      f2 = f(2)
      f3 = f(3)
      f4 = f(4)
      fd(1) = (w4*f1 + w1*f4 - w2*f3 + w3*f2)*0.5
      fd(2) = (w4*f2 + w1*f3 + w2*f4 - w3*f1)*0.5
      fd(3) = (w4*f3 - w1*f2 + w2*f1 + w3*f4)*0.5
      fd(4) = (w4*f4 - w1*f1 - w2*f2 - w3*f3)*0.5
      RETURN
      END
```

A.2.13 Gyro.for

```
      SUBROUTINE gyro(p, q, r, t, pulseg)
c-----------------------------------------------------------------------
c     function :            gyro model computes sensed delta angle
c                           counts. includes axis misalignment and
c                           nonorthogonality errors, scale factor
c                           errors,random and constant drift, and
c                           quantization.
c     inputs :              p,q,r,t
c     both :                pulseg
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL sfwig(3), sf1g(3), sf2g(3), qfracg(3), dcg(3)
      REAL sfeg(3), wdrg(3), dthet(3), pulseg(3)
      INTEGER gyseed
      DATA gyseed/123/, qfracg/3*0.0/, drmeng/0.0/, sppg/0.0/
      DATA sf1g, sf2g/6*0.0/, dcg/3*0.0/, psig, thtg, phig/3*0.0/
      DATA thxyg, thxzg, thyxg, thyzg, thzxg, thzyg/6*0.0/, drsigg/0.0/
#include "../include/constant.dat"
      pmm = p + q*psig - r*thtg
      qmm =  - p*psig + q + r*phig
      rmm = p*thtg - q*phig + r
      pn = pmm + qmm*thxzg - rmm*thxyg
      qn =  - pmm*thyzg + qmm + rmm*thyxg
      rn = pmm*thzyg - qmm*thzxg + rmm
      sfwig(1) = sf1g(1)*pn + sf2g(1)*pn**2
      sfwig(2) = sf1g(2)*qn + sf2g(2)*qn**2
      sfwig(3) = sf1g(3)*rn + sf2g(3)*rn**2
      sfeg(1) = pn + sfwig(1)
      sfeg(2) = qn + sfwig(2)
      sfeg(3) = rn + sfwig(3)
      IF (t .GT. 0.0) THEN
        DO 10 i = 1, 3
          CALL norm1(drsigg, drmeng, gyseed, drg)
          wdrg(i) = drg + dcg(i)
          dthet(i) = delt*(sfeg(i) + wdrg(i))
          IF (sppg .GT. 0.0) THEN
            qfracg(i) = qfracg(i) - pulseg(i) + dthet(i)/sppg
            pulseg(i) = aint(qfracg(i))
          ELSE
            pulseg(i) = dthet(i)
          ENDIF
   10   CONTINUE
      ELSE
        DO 20 i = 1, 3
          qfracg(i) = 0.0
          pulseg(i) = 0.0
   20   CONTINUE
      ENDIF
      RETURN
      END
```

A.2.14 Imupro.for

```
      SUBROUTINE imupro(pulsea, pulseg, delphi, deltht, delpsi, delu,
     & delv, delw)
c-----------------------------------------------------------------------
c     function :            computes the imu processor related functions
c     inputs :              pulsea,pulseg
c     outputs :             delu,delv,delw,delphi,deltht,delpsi
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL pulseg(3), pulsea(3)
      DATA sfcgx, sfcgy, sfcgz, sfcax, sfcay, sfcaz/6*1.0/
      DATA phigp/0.0/, thtgp/0.0/, psigp/0.0/
      DATA phiap/0.0/, thtap/0.0/, psiap/0.0/
      DATA perpg/0.0/, perpa/0.0/
      DATA iskull/0/
      IF (perpg .GT. 0.0) THEN
        delphs = pulseg(1)*perpg
        delths = pulseg(2)*perpg
        delpss = pulseg(3)*perpg
      ELSE
        delphs = pulseg(1)
        delths = pulseg(2)
        delpss = pulseg(3)
      ENDIF
      delph = delphs*sfcgx
      delth = delths*sfcgy
      delps = delpss*sfcgz
      delphi = delph - delth*psigp + delps*thtgp
      deltht = delph*psigp + delth - delps*phigp
      delpsi =  - delph*thtgp + delth*phigp + delps
      IF (perpa .GT. 0.0) THEN
        delus = pulsea(1)*perpa
        delvs = pulsea(2)*perpa
        delws = pulsea(3)*perpa
      ELSE
        delus = pulsea(1)
        delvs = pulsea(2)
        delws = pulsea(3)
      ENDIF
      delxs = delus*sfcax
      delys = delvs*sfcay
      delzs = delws*sfcaz
      delum = delxs - delys*psiap + delzs*thtap
      delvm = delxs*psiap + delys - delzs*phiap
      delwm =  - delxs*thtap + delys*phiap + delzs
      IF (iskull .EQ. 0) THEN
        delu = delum
        delv = delvm
        delw = delwm
      ELSE
        delu = delum - 0.5*(delpsi*delvm - deltht*delwm)
        delv = delvm - 0.5*(delphi*delwm - delpsi*delum)
        delw = delwm - 0.5*(deltht*delum - delphi*delvm)
      ENDIF
      RETURN
      END
```

A.2.15 Incorv.for

```
      SUBROUTINE incorv(vg, rmir, vmir, mvs, uvs)
c-----------------------------------------------------------------------
c     function :            to produce an initial estimate of correlated
c                           velocity
c     inputs :              rmir,vmir
c     outputs :             mvs,uvs
c     both :                vg
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION rb(3), vc(3), vphi(3), rtpred(3)
      DOUBLE PRECISION urb(3), urt(3), tmpv(3), utmpv(3), uthp(3)
      DOUBLE PRECISION vce(3), vse(3), rmir(3), vmir(3)
      DOUBLE PRECISION mrb, mrt, mtmpv, mvce, mvse
      REAL mvs, uvs(3), vg(3), vs(3), vd0(3)
      DATA rtpred/21227680.0, 0.0, 1168230.0/, vp1/13770.0/, ttf/140.0/
      DATA f1/450.0/, f2/-0.5/, vd0/3*0.0/, gmu/1.4052477e16/
      DO 10 i = 1, 3
        vce(i) = vg(i) + vmir(i)
        vse(i) = vg(i) - vd0(i)
   10 CONTINUE
      mvse = dsqrt(vse(1)**2 + vse(2)**2 + vse(3)**2)
      IF (mvse .GT. vp1) THEN
        scale3 = vp1/mvse
      ELSE
        scale3 = 1.0
      ENDIF
      scalar = f2*vp1*scale3/f1
      rb(1) = rmir(1) + scalar*vse(1)
      rb(2) = rmir(2) + scalar*vse(2)
      rb(3) = rmir(3) + scalar*vse(3)
      DO 30 ipass = 1, 50
        CALL magt(rb, mrb, urb)
        CALL magt(rtpred, mrt, urt)
        CALL crosst(urb, urt, tmpv)
        CALL magt(tmpv, mtmpv, utmpv)
        CALL crosst(utmpv, urb, uthp)
        vhc = vce(1)*uthp(1) + vce(2)*uthp(2) + vce(3)*uthp(3)
        vcr = vce(1)*urb(1) + vce(2)*urb(2) + vce(3)*urb(3)
        CALL crosst(urb, urt, vphi)
        sinphi = dsqrt(vphi(1)**2 + vphi(2)**2 + vphi(3)**2)
        cosphi = urb(1)*urt(1) + urb(2)*urt(2) + urb(3)*urt(3)
        mvce = dsqrt(vce(1)**2 + vce(2)**2 + vce(3)**2)
        w = vhc/mrb
        el = mrb*vhc**2/gmu
        ar = mrb/mrt
        tp1 = mvce**2*mrb/gmu
        hhh = el*sinphi**2*(2.0 - tp1)
        sqrhhh = dsqrt(hhh)
        t1 = el*sinphi/(hhh*w)
        t2a = (1.0 - el)/ar + 1.0 - ar*el
        t2b = (2.0*el - 1.0 - 1.0/ar)*cosphi
        t2 = t2a + t2b
        t3 = 2.0*el**2*sinphi**3/(w*hhh*sqrhhh)
        t4a = sqrhhh
        t4b = el + ar*el + cosphi - 1.0
        t4 = arctan(t4a, t4b)
        tffe = t1*t2 + t3*t4
        tff = ttf
        deltf = tff - tffe
        a = 2.0*(ar - cosphi)/sinphi + (vcr/vhc)
        b = a*vcr - vhc
```

```
      c = b*mrb/gmu
      d = c*el*sinphi**2
      e = d + hhh/vhc
      parhv = e*2.0
      part1v = (1.0/vhc - parhv/hhh)*t1
      part2v = (2.0*el/vhc)*(2.0*cosphi - (1.0 + ar**2)/ar)
      part3v = (1.0/vhc - parhv/(2.0*hhh))*3.0*t3
      subeq1 = (el + ar*el + cosphi - 1.0)*vhc*parhv
      subeq2 = 4.0*hhh*el*(1.0 + ar)
      subeq3 = (el + ar*el + cosphi - 1.0)**2 + hhh
      subeq4 = 2.0*sqrhhh*vhc
      part4v = (subeq1 - subeq2)/(subeq3*subeq4)
      ptffv = t1*part2v + t2*part1v + t3*part4v + t4*part3v
      vcopk = vhc + deltf/ptffv
      vcrpk = (vcopk/(el*sinphi))*(1.0 - ar*el - (1.0 - el)*cosphi)
      DO 20 j = 1, 3
        vc(j) = vcrpk*urb(j) + vcopk*uthp(j)
20    CONTINUE
30  CONTINUE
    DO 40 j = 1, 3
      vg(j) = vc(j) - vmir(j)
      vs(j) = vg(j) - vd0(j)
40  CONTINUE
    CALL spmagt(vs, mvs, uvs)
    RETURN
    END
```

A.2.16 Integ.for

```
      SUBROUTINE integi(x, xdot, t, i)
c-----------------------------------------------------------------------
c     function :            initialize integral of x which is stored
c                           in position i of the integral array
c     inputs :              x,xdot,t,i
c     outputs :             none
c-----------------------------------------------------------------------
      COMMON /storag/xint, tint, xdotl
      DOUBLE PRECISION xint(50), tint(50), xdotl(50)
      DOUBLE PRECISION x, t, xdot
      xint(i) = x
      xdotl(i) = xdot
      tint(i) = t
      RETURN
      END




      SUBROUTINE integ(x, xdot, t, i)
c-----------------------------------------------------------------------
c     function :            perform simple trapezoidal integration of
c                           xdot to yield x.  dtd is the time since
c                           the last integration and i is the array
c                           index where x is stored
c     inputs :              xdot,t,i
c     outputs :             x
c-----------------------------------------------------------------------
      COMMON /storag/xint, tint, xdotl
      DOUBLE PRECISION xint(50), tint(50), xdotl(50)
      DOUBLE PRECISION dt, dtmp, x
      DOUBLE PRECISION xdot, t
      dt = t - tint(i)
      xint(i) = xint(i) + 0.5d0*dt*(xdot + xdotl(i))
      x = xint(i)
      tint(i) = t
      xdotl(i) = xdot
      IF (i .EQ. 18) THEN
        dtmp = dsqrt(xint(15)**2 + xint(16)**2 + xint(17)**2 + xint(18)
     &    **2)
        xint(15) = xint(15)/dtmp
        xint(16) = xint(16)/dtmp
        xint(17) = xint(17)/dtmp
        xint(18) = xint(18)/dtmp
      ENDIF
      RETURN
      END
```

A.2.17 Magt.for

```
      SUBROUTINE magt(v, mv, uv)
C-------------------------------------------------------------------------
C     function :            computes the magnitude and unit vector of a
C                           given vector
C     inputs :              v
C     outputs :             mv, uv
C-------------------------------------------------------------------------
      DOUBLE PRECISION mv, v(3), uv(3)
      mv = dsqrt(v(1)**2 + v(2)**2 + v(3)**2)
      IF (mv .EQ. 0.0) THEN
        uv(1) = 0.0
        uv(2) = 0.0
        uv(3) = 0.0
      ELSE
        uv(1) = v(1)/mv
        uv(2) = v(2)/mv
        uv(3) = v(3)/mv
      ENDIF
      RETURN
      END
```

A.2.18 Missil.for

```
      SUBROUTINE missil(t, mass, fxt, frcx, fyt, frcy, fzt, frcz, xyz,
     & xyzd, ud, vd, wd, gr, cim, xyzdd)
c-----------------------------------------------------------------------
c     function :            computes the rotational and translational
c                           missile accelerations
c     inputs :              t,mass,fxa,fxt,frcx,
c                           fya,fyt,frcy,fza,fzt,frcz,mrcx,
c                           mrcy,mrcz,xyz,xyzd
c     outputs :             ud,vd,wd,gr,cim,xyzdd
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      REAL gr(3)
      DOUBLE PRECISION gb(3), xyzlch(3)
      DOUBLE PRECISION xyz(3), xyzd(3), uxyz(3)
      DOUBLE PRECISION xyzdd(3)
      DOUBLE PRECISION mgr, mxyz
      REAL mass, frcx, frcy, frcz
      REAL fxt, fyt, fzt, fxa, fya, fza
      REAL cmi(9), cim(9), ud, vd, wd
      DATA nclear/0/, imis/0/, rade/20898908.0/, xlnch/3.0/
      DATA gmu/1.4052477e16/
      IF (imis .EQ. 0) THEN
        CALL sptrans(cim, cmi)
        xyzlch(1) = xlnch*cmi(1) + rade
        xyzlch(2) = xlnch*cmi(2)
        xyzlch(3) = xlnch*cmi(3)
        imis = 1
      ENDIF
      CALL magt(xyz, mxyz, uxyz)
      mgr = gmu/mxyz**2
c      CALL mvbys(-mgr, uxyz, gr)
      gr(1) = -mgr*uxyz(1)
      gr(2) = -mgr*uxyz(2)
      gr(3) = -mgr*uxyz(3)

c           CALL vecrot(gr, cim, gb)
      gb(1) = cim(1)*gr(1) + cim(4)*gr(2) + cim(7)*gr(3)
      gb(2) = cim(2)*gr(1) + cim(5)*gr(2) + cim(8)*gr(3)
      gb(3) = cim(3)*gr(1) + cim(6)*gr(2) + cim(9)*gr(3)

      CALL receive_real_32bit(fxa)
      CALL receive_real_32bit(fya)
      CALL receive_real_32bit(fza)
      fx = fxt + fxa + frcx
      fy = fyt + fya + frcy
      fz = fzt + fza + frcz
      IF (nclear .EQ. 1) THEN
        ud = fx/mass + gb(1)
        vd = fy/mass + gb(2)
        wd = fz/mass + gb(3)
      ELSEIF (fx/mass .LE. dabs(gb(1))) THEN
        gb(1) = 0.0
        gb(2) = 0.0
        gb(3) = 0.0
c-----------   call vecrot(gb,cmi,gr)
        gr(1) = cim(1)*gb(1) + cim(2)*gb(2) + cim(3)*gb(3)
        gr(2) = cim(4)*gb(1) + cim(5)*gb(2) + cim(6)*gb(3)
        gr(3) = cim(7)*gb(1) + cim(8)*gb(2) + cim(9)*gb(3)
        ud = 0.0
        vd = 0.0
        wd = 0.0
```

```
      ELSEIF (xyz(1) .LE. xyzlch(1) .AND. xyz(2) .LE. xyzlch(2) .AND.
     & xyz(3) .LE. xyzlch(3)) THEN
        gb(2) = 0.0
        gb(3) = 0.0
c-----------   call vecrot(gb,cmi,gr)
        gr(1) = cim(1)*gb(1) + cim(2)*gb(2) + cim(3)*gb(3)
        gr(2) = cim(4)*gb(1) + cim(5)*gb(2) + cim(6)*gb(3)
        gr(3) = cim(7)*gb(1) + cim(8)*gb(2) + cim(9)*gb(3)
        ud = fx/mass + gb(1)
        vd = 0.0
        wd = 0.0
      ELSE
        nclear = 1
        ud = fx/mass + gb(1)
        vd = fy/mass + gb(2)
        wd = fz/mass + gb(3)
      ENDIF
      xyzdd(1) = cim(1)*ud + cim(2)*vd + cim(3)*wd
      xyzdd(2) = cim(4)*ud + cim(5)*vd + cim(6)*wd
      xyzdd(3) = cim(7)*ud + cim(8)*vd + cim(9)*wd
      RETURN
      END
```

A.2.19 Mmk.for

```
      SUBROUTINE mmk(a, na, b, nb, c, nc, rm)
c-----------------------------------------------------------------------
c     function  :           generates a direction cosine matrix
c                           by rotating in order:
c                             1) angle c about the nc axis
c                             2) angle b about the nb axis
c                             3) angle a about the na axis
c     inputs :              a,na,b,nb,c,nc
c     outputs :             rm
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DIMENSION am(3, 3), bm(3, 3), cm(3, 3), rm(3, 3), t(9)
      CALL rotmx(a, na, am)
      CALL rotmx(b, nb, bm)
      CALL rotmx(c, nc, cm)
      CALL mmlxy(bm, cm, t)
      CALL mmlxy(am, t, rm)
      RETURN
      END
```

A.2.20 Mmlxy.for

```
      SUBROUTINE mmlxy(x, y, z)
c-----------------------------------------------------------------------
c     function  :           multiply two 3x3 matrices
c     inputs :              x, y
c     outputs :             z
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DIMENSION x(3, 3), y(3, 3), z(3, 3)
      z(1, 1) = x(1, 1)*y(1, 1) + x(1, 2)*y(2, 1) + x(1, 3)*y(3, 1)
      z(2, 1) = x(2, 1)*y(1, 1) + x(2, 2)*y(2, 1) + x(2, 3)*y(3, 1)
      z(3, 1) = x(3, 1)*y(1, 1) + x(3, 2)*y(2, 1) + x(3, 3)*y(3, 1)
      z(1, 2) = x(1, 1)*y(1, 2) + x(1, 2)*y(2, 2) + x(1, 3)*y(3, 2)
      z(2, 2) = x(2, 1)*y(1, 2) + x(2, 2)*y(2, 2) + x(2, 3)*y(3, 2)
      z(3, 2) = x(3, 1)*y(1, 2) + x(3, 2)*y(2, 2) + x(3, 3)*y(3, 2)
      z(1, 3) = x(1, 1)*y(1, 3) + x(1, 2)*y(2, 3) + x(1, 3)*y(3, 3)
      z(2, 3) = x(2, 1)*y(1, 3) + x(2, 2)*y(2, 3) + x(2, 3)*y(3, 3)
      z(3, 3) = x(3, 1)*y(1, 3) + x(3, 2)*y(2, 3) + x(3, 3)*y(3, 3)
      RETURN
      END
```

A.2.21 Mvbys.for

```
      SUBROUTINE mvbys(a, b, c)
c-----------------------------------------------------------------------
c     function  :           multiplies a vector by a scalar value
c     inputs :              a, b
c     outputs :             c
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DIMENSION b(3), c(3)
      c(1) = a*b(1)
      c(2) = a*b(2)
      c(3) = a*b(3)
      RETURN
      END
```

A.2.22 Navig.for

```
      SUBROUTINE navig(delphi, deltht, delpsi, delu, delv, delw, gr, t,
     & sq, sr, ti2m, at, delxd, delyd, delzd)
c-----------------------------------------------------------------------
c     function :            computes the quaternions and transformation
c                           matrices using delta angles sensed by the
c                           gyro.computes the position and velocity in
c                           inertial and earth-centered frames.
c                           computes sensed body rates, euler angles
c                           and the gravity-compensated acceleration.
c     inputs :              delphi,deltht,delpsi,delu,delv,delw,gr,t
c     outputs :             ti2m,at
c     both :                sq,sr,vmir,rmir
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL ti2m(9), tm2i(9)
      REAL gr(3), at(3)
      REAL atg(3), temp(3), qs1(4)
      REAL sq, sr, delphi, deltht, delpsi, delu, delv, delw
      DATA mnav/0/, dtx0/0.0/, dty0/0.0/, dtz0/0.0/
      DATA sphiic/0.0/, sthtic/-35.0/, spsiic/0.0/
      DATA dtr/0.017453292519943296/
#include "../include/constant.dat"
      IF (mnav .EQ. 0) THEN
        sith0 = sin(sthtic*dtr/2.0)
        coth0 = cos(sthtic*dtr/2.0)
        sips0 = sin(spsiic*dtr/2.0)
        cops0 = cos(spsiic*dtr/2.0)
        siph0 = sin(sphiic*dtr/2.0)
        coph0 = cos(sphiic*dtr/2.0)
        qs1(4) = cops0*coth0*coph0 + sips0*sith0*siph0
        qs1(1) = cops0*coth0*siph0 - sips0*sith0*coph0
        qs1(2) = cops0*sith0*coph0 + sips0*coth0*siph0
        qs1(3) =  - cops0*sith0*siph0 + sips0*coth0*coph0
      ENDIF
      dtx = 0.5e0*delphi
      dty = 0.5e0*deltht
      dtz = 0.5e0*delpsi
      pp0 = dtx**2 + dty**2 + dtz**2
      pp1 = (pp0*dtx + dty*dtz0 - dtz*dty0)/6.0e0
      pp2 = (pp0*dty + dtz*dtx0 - dtx*dtz0)/6.0e0
      pp3 = (pp0*dtz + dtx*dty0 - dty*dtx0)/6.0e0
      dtx0 = dtx
      dty0 = dty
      dtz0 = dtz
      dtx = dtx - pp1
      dty = dty - pp2
      dtz = dtz - pp3
      dum =  - 0.5e0*pp0
      pq0 = dum*qs1(4) - dtx*qs1(1) - dty*qs1(2) - dtz*qs1(3)
      pq1 = dtx*qs1(4) + dum*qs1(1) + dtz*qs1(2) - dty*qs1(3)
      pq2 = dty*qs1(4) - dtz*qs1(1) + dum*qs1(2) + dtx*qs1(3)
      pq3 = dtz*qs1(4) + dty*qs1(1) - dtx*qs1(2) + dum*qs1(3)
      qs1(4) = qs1(4) + pq0
      qs1(1) = qs1(1) + pq1
      qs1(2) = qs1(2) + pq2
      qs1(3) = qs1(3) + pq3
      IF (mnav .GE. 100) THEN
        dq = 0.5e0*(1.0e0 - qs1(4)**2 - qs1(1)**2 - qs1(2)**2 - qs1(3)*
     &   *2)
        qs1(1) = qs1(1)*(1.0e0 + dq)
        qs1(2) = qs1(2)*(1.0e0 + dq)
```

```
        qs1(3) = qs1(3)*(1.0e0 + dq)
        qs1(4) = qs1(4)*(1.0e0 + dq)
        mnav = 1
      ELSE
        mnav = mnav + 1
      ENDIF
      ti2m(1) = qs1(4)**2 + qs1(1)**2 - qs1(2)**2 - qs1(3)**2
      ti2m(2) = 2.0e0*(qs1(1)*qs1(2) - qs1(4)*qs1(3))
      ti2m(3) = 2.0e0*(qs1(1)*qs1(3) + qs1(4)*qs1(2))
      ti2m(4) = 2.0e0*(qs1(1)*qs1(2) + qs1(4)*qs1(3))
      ti2m(5) = qs1(4)**2 - qs1(1)**2 + qs1(2)**2 - qs1(3)**2
      ti2m(6) = 2.0e0*(qs1(2)*qs1(3) - qs1(4)*qs1(1))
      ti2m(7) = 2.0e0*(qs1(1)*qs1(3) - qs1(4)*qs1(2))
      ti2m(8) = 2.0e0*(qs1(2)*qs1(3) + qs1(4)*qs1(1))
      ti2m(9) = qs1(4)**2 - qs1(1)**2 - qs1(2)**2 + qs1(3)**2
c      call trans(ti2m,tm2i)
      IF (delt .GT. 0.0) THEN
        sq = deltht/delt
        sr = delpsi/delt
        sud = delu/delt
        svd = delv/delt
        swd = delw/delt
      ENDIF
      CALL vmk(sud, svd, swd, temp)
c      call vecrot(temp,tm2i,atg)
      atg(1) = ti2m(1)*temp(1) + ti2m(2)*temp(2) + ti2m(3)*temp(3)
      atg(2) = ti2m(4)*temp(1) + ti2m(5)*temp(2) + ti2m(6)*temp(3)
      atg(3) = ti2m(7)*temp(1) + ti2m(8)*temp(2) + ti2m(9)*temp(3)
      CALL spvecsub(atg, gr, at)
      delxd = ti2m(1)*delu + ti2m(2)*delv + ti2m(3)*delw
      delyd = ti2m(4)*delu + ti2m(5)*delv + ti2m(6)*delw
      delzd = ti2m(7)*delu + ti2m(8)*delv + ti2m(9)*delw
      RETURN
      END
```

A.2.23 Ncu.for

```
      SUBROUTINE ncu(dlp, dly, cmmd, dlpd, dlyd)
c-----------------------------------------------------------------------
c     function :            models the response of the nozzle
c                           control unit
c     inputs :              dlp,dly,cmmd
c     outputs :             dlpd,dlyd
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL kncu
      REAL cmmd(2)
      DATA kncu/1./, omegat/66.66667/, rmax/1.047198/
      dlpd = (cmmd(1) - kncu*dlp)*omegat
      dlyd = (cmmd(2) - kncu*dly)*omegat
      totrat = sqrt(dlpd**2 + dlyd**2)
      IF (totrat .GT. rmax) THEN
        dlpd = dlpd*rmax/totrat
        dlyd = dlyd*rmax/totrat
      ENDIF
      RETURN
      END
```

A.2.24 Norm1.for

```
      SUBROUTINE norm1(sd, mn, iseed, rdn)
c-----------------------------------------------------------------------
c      function :            generates normally distributed random
c                            numbers using the hastings approximation
c      inputs :              sd,mn
c      outputs :             rdn
c      both :                iseed
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h. o-z)
      REAL mn
      REAL ran
      DATA c0, c1, c2/2.515517, 0.802853, 0.010328/
      DATA d1, d2, d3/1.432788, 0.189269, 0.001308/
      pu1 = ran(iseed)
      IF (pu1 .EQ. 0.5) pu1 = ran(iseed)
      pu1 = pu1 - 0.5
      pu = abs(pu1)
      tn = sqrt(abs(2.0*log(pu)))
      an = c0 + c1*tn + c2*(tn**2)
      b = 1.0 + d1*tn + d2*(tn**2) + d3*(tn**3)
      xn = tn - (an/b) - 1.0e-5
      IF (pu1 .LT. 0.0) xn =  - xn
      rdn = xn*sd + mn
      RETURN
      END
```

A.2.25 Qntzr.for

```
      DOUBLE PRECISION FUNCTION qntzr(a, b)
c-------------------------------------------------------------------------
c      function :            quantize the input variable b
c      inputs :              a,b
c      outputs :             qntzr
c-------------------------------------------------------------------------
      DOUBLE PRECISION a, b
      IF (a .GT. 0.0d0) THEN
        qntzr = (dint(b)/a + 0.5d0)*a
      ELSE
        qntzr = b
      ENDIF
      END
```

A.2.26 Ran.for

```
      REAL FUNCTION ran(iseed)
c-----------------------------------------------------------------------
c     function  :           random number
c     called from  :        utility subroutine
c     subroutines called :  none
c     inputs :              iseed
c     outputs :             ran
c-----------------------------------------------------------------------
      iseed = 69069*iseed + 1
      ran = abs(float(iseed)/2147483647.0)
      RETURN
      END
```

A.2.27 Rotmx.for

```
      SUBROUTINE rotmx(x, i, xm)
c-----------------------------------------------------------------------
c     function  :           generates a direction cosine matrix
c     inputs :              x, i
c     outputs :             xm
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION xm(3, 3)
      INTEGER iit(3), iiit(3)
      DATA iit/2, 3, 1/, iiit/3, 1, 2/
      sx = dsin(x
      cx = dcos(x)
      ii = iit(i)
      iii = iiit(i)
      xm(i, i) = 1.0
      xm(i, ii) = 0.0
      xm(i, iii) = 0.0
      xm(ii, i) = 0.0
      xm(iii, i) = 0.0
      xm(ii, ii) = cx
      xm(iii, iii) = cx
      xm(ii, iii) = sx
      xm(iii, ii) =  - sx
      RETURN
      END
```

A.2.28 Spcrosst.for

```
      SUBROUTINE spcrosst(v1, v2, r)
c-----------------------------------------------------------------------
c     function :            takes the cross product of two vectors
c
c                           ( v1 x v2 ) = r
c     inputs :              v1, v2
c     outputs :             r
c-----------------------------------------------------------------------
      REAL v1(3), v2(3), r(3)
      r(1) = v1(2)*v2(3) - v1(3)*v2(2)
      r(2) = v1(3)*v2(1) - v1(1)*v2(3)
      r(3) = v1(1)*v2(2) - v1(2)*v2(1)
      RETURN
      END
```

A.2.29 Spinteg.for

```
      SUBROUTINE spintegi(x, xdot, t, i)
c-----------------------------------------------------------------------
c     function :            initialize integral of x which is stored
c                           in position i of the integral array
c     inputs :              x,xdot,t,i
c     outputs :             none
c-----------------------------------------------------------------------
      COMMON /spstorag/xint, tint, xdotl
      REAL xint(50), tint(50), xdotl(50)
      REAL t, xdot
      PEAL x
      xint(i) = x
      xdotl(i) = xdot
      tint(i) = t
      RETURN
      END



      SUBROUTINE spinteg(x, xdot, t, i)
c-----------------------------------------------------------------------
c     function :            perform simple trapezoidal integration of
c                           xdot to yield x.  dtd is the time since
c                           the last integration and i is the array
c                           index where x is stored
c     inputs :              xdot,t,i
c     outputs :             x
c-----------------------------------------------------------------------
      COMMON /spstorag/xint, tint, xdotl
      REAL xint(50), tint(50), xdotl(50)
      REAL dt, dtmp
      REAL xdot, t
      REAL x
      dt = t - tint(i)
      xint(i) = xint(i) + 0.5e0*dt*(xdot + xdotl(i))
      x = xint(i)
      tint(i) = t
      xdotl(i) = xdot
      IF (i .EQ. 18) THEN
        dtmp = sqrt(xint(15)**2 + xint(16)**2 + xint(17)**2 + xint(18)*
     &    *2)
        xint(15) = xint(15)/dtmp
        xint(16) = xint(16)/dtmp
        xint(17) = xint(17)/dtmp
        xint(18) = xint(18)/dtmp
      ENDIF
      RETURN
      END
```

A.2.30 Spmagt.for

```
      SUBROUTINE spmagt(v, mv, uv)
c-------------------------------------------------------------------------
c     function :            computes the magnitude and unit vector of a
c                           given vector
c     inputs :              v
c     outputs :             mv, uv
c-------------------------------------------------------------------------
      REAL mv, v(3), uv(3)
      mv = sqrt(v(1)**2 + v(2)**2 + v(3)**2)
      IF (mv .EQ. 0.0) THEN
        uv(1) = 0.0
        uv(2) = 0.0
        uv(3) = 0.0
      ELSE
        uv(1) = v(1)/mv
        uv(2) = v(2)/mv
        uv(3) = v(3)/mv
      ENDIF
      RETURN
      END
```

A.2.31 Spmmk.for

```
      SUBROUTINE spmmk(a, na, b, nb, c, nc, rm)
c-----------------------------------------------------------------------
c     function  :           generates a direction cosine matrix
c                           by rotating in order:
c                             1) angle c about the nc axis
c                             2) angle b about the nb axis
c                             3) angle a about the na axis
c     inputs :              a,na,b,nb,c,nc
c     outputs :             rm
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      DIMENSION am(3, 3), bm(3, 3), cm(3, 3), rm(3, 3), t(9)
      CALL sprotmx(a, na, am)
      CALL sprotmx(b, nb, bm)
      CALL sprotmx(c, nc, cm)
      CALL spmmlxy(bm, cm, t)
      CALL spmmlxy(am, t, rm)
      RETURN
      END
```

A.2.32 Spmmlxy.for

```
      SUBROUTINE spmmlxy(x, y, z)
c-----------------------------------------------------------------------
c     function  :           multiply two 3x3 matrices
c     inputs :              x, y
c     outputs :             z
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      DIMENSION x(3, 3), y(3, 3), z(3, 3)
      z(1, 1) = x(1, 1)*y(1, 1) + x(1, 2)*y(2, 1) + x(1, 3)*y(3, 1)
      z(2, 1) = x(2, 1)*y(1, 1) + x(2, 2)*y(2, 1) + x(2, 3)*y(3, 1)
      z(3, 1) = x(3, 1)*y(1, 1) + x(3, 2)*y(2, 1) + x(3, 3)*y(3, 1)
      z(1, 2) = x(1, 1)*y(1, 2) + x(1, 2)*y(2, 2) + x(1, 3)*y(3, 2)
      z(2, 2) = x(2, 1)*y(1, 2) + x(2, 2)*y(2, 2) + x(2, 3)*y(3, 2)
      z(3, 2) = x(3, 1)*y(1, 2) + x(3, 2)*y(2, 2) + x(3, 3)*y(3, 2)
      z(1, 3) = x(1, 1)*y(1, 3) + x(1, 2)*y(2, 3) + x(1, 3)*y(3, 3)
      z(2, 3) = x(2, 1)*y(1, 3) + x(2, 2)*y(2, 3) + x(2, 3)*y(3, 3)
      z(3, 3) = x(3, 1)*y(1, 3) + x(3, 2)*y(2, 3) + x(3, 3)*y(3, 3)
      RETURN
      END
```

A.2.33 Spmvbys.for

```
      SUBROUTINE spmvbys(a, b, c)
c-----------------------------------------------------------------------
c     function  :           multiplies a vector by a scalar value
c     inputs :              a, b
c     outputs :             c
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      DIMENSION b(3), c(3)
      c(1) = a*b(1)
      c(2) = a*b(2)
      c(3) = a*b(3)
      RETURN
      END
```

A.2.34 Sprotmx.for

```
      SUBROUTINE sprotmx(x, i, xm)
c-----------------------------------------------------------------------
c     function  :           generates a direction cosine matrix
c     inputs :              x,i
c     outputs :             xm
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL xm(3, 3)
      INTEGER iit(3), iiit(3)
      DATA iit/2, 3, 1/, iiit/3, 1, 2/
      sx = sin(x)
      cx = cos(x)
      ii = iit(i)
      iii = iiit(i)
      xm(i, i) = 1.0
      xm(i, ii) = 0.0
      xm(i, iii) = 0.0
      xm(ii, i) = 0.0
      xm(iii, i) = 0.0
      xm(ii, ii) = cx
      xm(iii, iii) = cx
      xm(ii, iii) = sx
      xm(iii, ii) =  - sx
      RETURN
      END
```

A.2.35 Sptrans.for

```
      SUBROUTINE sptrans(a, b)
c-----------------------------------------------------------------------
c     function  :           3x3 matrix transposition
c     inputs :              a
c     outputs :             b
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL a(9), b(9)
      b(1) = a(1)
      b(2) = a(4)
      b(3) = a(7)
      b(4) = a(2)
      b(5) = a(5)
      b(6) = a(8)
      b(7) = a(3)
      b(8) = a(6)
      b(9) = a(9)
      RETURN
      END
```

A.2.36 Spvecrot.for

```
      SUBROUTINE spvecrot(vin, rmx, vout)
c-----------------------------------------------------------------------
c     function  :           rotate a vector from one coordinate
c                           frame to another through rotation
c                           matrix rmx.
c     inputs :              vin, rmx
c     outputs :             vout
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL vin(3), rmx(3, 3), vout(3)
      vout(1) = rmx(1, 1)*vin(1) + rmx(1, 2)*vin(2) + rmx(1, 3)*vin(3)
      vout(2) = rmx(2, 1)*vin(1) + rmx(2, 2)*vin(2) + rmx(2, 3)*vin(3)
      vout(3) = rmx(3, 1)*vin(1) + rmx(3, 2)*vin(2) + rmx(3, 3)*vin(3)
      RETURN
      END
```

A.2.37 Spvecsub.for

```
      SUBROUTINE spvecsub(v1, v2, r)
c-----------------------------------------------------------------------
c     function  :           subtract two 3x1 vectors
c
c                           ( v1 - v2 ) = r
c     inputs :              v1, v2
c     outputs :             r
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL v1(3), v2(3), r(3)
      r(1) = v1(1) - v2(1)
      r(2) = v1(2) - v2(2)
      r(3) = v1(3) - v2(3)
      RETURN
      END
```

A.2.38 Table.for

```
      SUBROUTINE table(xtab, ytab, x, y, n, i)
c-----------------------------------------------------------------------
c      function :            performs table lookup via either indexed
c                            search or binary search and linearly
c                            interpolates
c      inputs :              xtab,ytab,x,n
c      outputs :             y
c      both :                i
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      INTEGER n, i
      REAL xtab(n), ytab(n)
      IF (i .GE. 1 .AND. i .LE. n) THEN
        IF (x .LE. xtab(1)) THEN
          y = ytab(1)
          i = 1
        ELSEIF (x .GE. xtab(n)) THEN
          y = ytab(n)
          i = n
        ELSEIF (x .GE. xtab(i)) THEN
          DO 10 k = i, n - 1
            IF (x .LT. xtab(k + 1)) GOTO 20
   10     CONTINUE
   20     fract = (x - xtab(k))/(xtab(k + 1) - xtab(k))
          y = ytab(k) + fract*(ytab(k + 1) - ytab(k))
          i = k
        ELSEIF (x .LT. xtab(i)) THEN
          DO 30 k = i - 1, 1, -1
            IF (x .GE. xtab(k)) GOTO 40
   30     CONTINUE
   40     fract = (x - xtab(k))/(xtab(k + 1) - xtab(k))
          y = ytab(k) + fract*(ytab(k + 1) - ytab(k))
          i = k
        ENDIF
      ELSEIF (i .LT. 1 .OR. i .GT. n) THEN
        IF (x .GT. xtab(1) .AND. x .LT. xtab(n)) THEN
          k = 1
          l = n
          DO 50 i = k, 1
            IF (l .EQ. k + 1) GOTO 60
            m = (k + l)/2
            IF (x .LT. xtab(m)) THEN
              l = m
            ELSE
              k = m
            ENDIF
   50     CONTINUE
   60     fract = (x - xtab(k))/(xtab(l) - xtab(k))
          y = ytab(k) + fract*(ytab(l) - ytab(k))
          i = k
        ELSEIF (x .LE. xtab(1)) THEN
          y = ytab(1)
          i = 1
        ELSEIF (x .GE. xtab(n)) THEN
          y = ytab(n)
          i = n
        ENDIF
      ENDIF
      RETURN
      END
```

A.2.39 Tlu2ei.for

```
      SUBROUTINE tlu2ei(x, y, f, i, j, tbval)
c-----------------------------------------------------------------------
c      function :            performs a linear table look-up of a table
c                            with two independent variables, and returns
c                            indices pointing to the area of the table
c                            in use
c      inputs :              x,y,f
c      outputs :             i,j,tbval
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL f(3)
      EQUIVALENCE (n12, nyu), (n21, nxl), (n22, nxu), (n11, isp)
      EQUIVALENCE (dx, xx), (dy, yy)
c  compute upper and lower bounds on indices for xx and yy
      nxu = abs(f(1)) + .1
      mpl = abs(f(2)) + 1.1
      nyu = mpl + 1
      nxl = nyu + 1
      nxu = nxu*mpl + 2
      js = j
      is = i
      xx = x
      yy = y
      IF (.NOT. ((f(1) .GE. 0.0) .AND. (f(2) .GE. 0.0))) THEN
        xx = y
        yy = x
      ENDIF
      IF (is .LT. nxl) is = nxl
      IF (js .LT. 3) js = 3
   10 CONTINUE
      jsp = js + 1
      IF (yy .GT. f(jsp)) THEN
        IF (jsp .EQ. nyu) GOTO 30
        js = jsp
        GOTO 10
      ENDIF
   20 IF (yy .LT. f(js)) THEN
        IF (js .NE. 3) THEN
          js = js - 1
          GOTO 20
        ENDIF
      ENDIF
   30 CONTINUE
      isp = is + mpl
      IF (xx .GT. f(isp)) THEN
        IF (isp .EQ. nxu) GOTO 50
        is = isp
        GOTO 30
      ENDIF
   40 IF (xx .LT. f(is)) THEN
        IF (is .NE. nxl) THEN
          is = is - mpl
          GOTO 40
        ENDIF
      ENDIF
   50 CONTINUE
      n11 = is + js - 2
      n12 = n11 + 1
      n21 = n11 + mpl
      n22 = n21 + 1
      ipmpl = is + mpl
```

```
dx = (xx - f(is))/(f(ipmp1) - f(is))
xj = (f(n21) - f(n11))*dx + f(n11)
xjp1 = (f(n22) - f(n12))*dx + f(n12)
dy = (yy - f(js))/(f(js + 1) - f(js))
j = js
i = is
tbval = (xjp1 - xj)*dy + xj
RETURN
END
```

A.2.40 Trans.for

```
      SUBROUTINE trans(a, b)
c-----------------------------------------------------------------------
c     function  :           3x3 matrix transposition
c     inputs :              a
c     outputs :             b
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION a(9), b(9)
      b(1) = a(1)
      b(2) = a(4)
      b(3) = a(7)
      b(4) = a(2)
      b(5) = a(5)
      b(6) = a(8)
      b(7) = a(3)
      b(8) = a(6)
      b(9) = a(9)
      RETURN
      END
```

A.2.41 Vecadd.for

```
      SUBROUTINE vecadd(v1, v2, r)
c-----------------------------------------------------------------------
c     function  :           add two 3x1 vectors
c
c                           ( v1 + v2 ) = r
c     inputs :              v1, v2
c     outputs :             r
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION v1(3), v2(3), r(3)
      r(1) = v1(1) + v2(1)
      r(2) = v1(2) + v2(2)
      r(3) = v1(3) + v2(3)
      RETURN
      END
```

A.2.42 Vecrot.for

```
      SUBROUTINE vecrot(vin, rmx, vout)
c-----------------------------------------------------------------------
c     function  :           rotate a vector from one coordinate
c                           frame to another through rotation
c                           matrix rmx.
c     inputs :              vin, rmx
c     outputs :             vout
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION vin(3), rmx(3, 3), vout(3)
      vout(1) = rmx(1, 1)*vin(1) + rmx(1, 2)*vin(2) + rmx(1, 3)*vin(3)
      vout(2) = rmx(2, 1)*vin(1) + rmx(2, 2)*vin(2) + rmx(2, 3)*vin(3)
      vout(3) = rmx(3, 1)*vin(1) + rmx(3, 2)*vin(2) + rmx(3, 3)*vin(3)
      RETURN
      END
```

A.2.43 Vecsub.for

```
      SUBROUTINE vecsub(v1, v2, r)
c-----------------------------------------------------------------------
c     function  :           subtract two 3x1 vectors
c
c                           ( v1 - v2 ) = r
c     inputs :              v1, v2
c     outputs :             r
c-----------------------------------------------------------------------
      IMPLICIT DOUBLEPRECISION(a-h, o-z)
      DOUBLE PRECISION v1(3), v2(3), r(3)
      r(1) = v1(1) - v2(1)
      r(2) = v1(2) - v2(2)
      r(3) = v1(3) - v2(3)
      RETURN
      END
```

A.2.44 Vmk.for

```
      SUBROUTINE vmk(x1, x2, x3, v)
c-----------------------------------------------------------------------
c     function  :           generates a vector from three components
c     inputs :              x1, x2, x3
c     outputs :             v
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL v(3)
      v(1) = x1
      v(2) = x2
      v(3) = x3
      RETURN
      END
```

A.2.45 Xferb.for

```
      SUBROUTINE xferb(a, i, b)
c-----------------------------------------------------------------------
c     function  :           vector move
c     inputs :              a,i
c     outputs :             b
c-----------------------------------------------------------------------
      IMPLICIT REAL(a-h, o-z)
      REAL a(i), b(i)
      DO 10 j = 1, i
        b(j) = a(j)
   10 CONTINUE
      RETURN
      END
```

A.3 Mainlines (C)

A.3.1 Aeroca.c

```
/* aeroca.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static integer icaa1 = 0;
    static integer icam2 = 0;
    static integer icaa2 = 0;
    static real tfinal = (float)62.501;
    static real tstg1 = (float)23.;
    static real calm[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)
          160.,(float)170.,(float)180.,(float)0.,(float)-.166,(float)-
.169,(
          float)-.161,(float)-.161,(float)-.15,(float)-.116,(float)-
.115,(
          float)-.031,(float)-
.004,(float)0.,(float).011,(float).076,(float)
          .153,(float).23,(float).243,(float).254,(float).5,(float)-
.176,(
          float)-.156,(float)-.15,(float)-.167,(float)-.126,(float)-
.116,(
          float)-.112,(float)-.048,(float)-.003,(float)0.,(float).007,(

float).06,(float).142,(float).229,(float).238,(float).239,(float)
          .8,(float)-.179,(float)-.196,(float)-.168,(float)-.162,(float)
          -.152,(float)-.136,(float)-.103,(float)-.037,(float)-
.005,(float)

0.,(float).022,(float).082,(float).194,(float).297,(float).32,(
          float).329,(float).9,(float)-.225,(float)-.21,(float)-
.219,(float)
          -.228,(float)-.189,(float)-.163,(float)-.121,(float)-
.051,(float)
          -
.015,(float)0.,(float).004,(float).091,(float).232,(float).341,(
          float).387,(float).391,(float)1.1,(float)-.35,(float)-
.34,(float)
          -.336,(float)-.343,(float)-.315,(float)-.281,(float)-
.218,(float)
          -.09,(float)-
.019,(float)0.,(float).015,(float).132,(float).31,(
          float).482,(float).522,(float).52,(float)1.2,(float)-
.334,(float)
          -.351,(float)-.348,(float)-.328,(float)-.318,(float)-
.276,(float)
```

```
          -.201,(float)-.066,(float)-
.014,(float)0.,(float).034,(float).169,

(float).418,(float).633,(float).694,(float).716,(float)1.5,(float)
          -.315,(float)-.308,(float)-.309,(float)-.29,(float)-
.28,(float)
          -.244,(float)-.188,(float)-.08,(float)-
.007,(float)0.,(float).014,
          (float).156,(float).381,(float).564,(float).631,(float).646,(
          float)2.,(float)-.267,(float)-.252,(float)-.284,(float)-.285,(
          float)-.35,(float)-.389,(float)-.433,(float)-.363,(float)-
.202,(
          float)-
.11,(float)0.,(float).204,(float).385,(float).504,(float)
          .557,(float).582,(float)4.,(float)-.168,(float)-.173,(float)-
.217,
          (float)-.231,(float)-.236,(float)-.298,(float)-.369,(float)-
.455,(
          float)-.475,(float)-.471,(float).359,(float).125,(float).31,(
          float).483,(float).514,(float).532,(float)6.,(float)-
.136,(float)
          -.128,(float)-.178,(float)-.184,(float)-.224,(float)-
.267,(float)
          -.342,(float)-.443,(float)-.46,(float)-
.446,(float).346,(float)

.045,(float).293,(float).425,(float).482,(float).485,(float)9.,(
          float)-.156,(float)-.136,(float)-.158,(float)-.17,(float)-
.203,(
          float)-.263,(float)-.323,(float)-.428,(float)-.483,(float)-
.449,(

float).351,(float).036,(float).281,(float).422,(float).467,(float)
          .501 };
    static real ca2m[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)
          160.,(float)170.,(float)180.,(float)0.,(float)-.106,(float)-
.115,(
          float)-.113,(float)-.1,(float)-.074,(float)-.077,(float)-
.053,(
          float)-.019,(float)-
.005,(float)0.,(float).019,(float).046,(float)
          .099,(float).147,(float).169,(float).166,(float).5,(float)-
.108,(
          float)-.128,(float)-.109,(float)-.097,(float)-.078,(float)-
.079,(
          float)-.077,(float)-.013,(float)-.003,(float)0.,(float).002,(

float).032,(float).094,(float).136,(float).171,(float).186,(float)
          .8,(float)-.122,(float)-.113,(float)-.119,(float)-.11,(float)
          -.099,(float)-.088,(float)-.091,(float)-.03,(float)-
.002,(float)

0.,(float).019,(float).061,(float).137,(float).203,(float).218,(
          float).222,(float).9,(float)-.166,(float)-.157,(float)-.153,(
          float)-.151,(float)-.117,(float)-.124,(float)-.109,(float)-
.034,(
          float)-
.011,(float)0.,(float).012,(float).052,(float).145,(float)
```

```
          .187,(float).242,(float).253,(float)1.1,(float)-.299,(float)-
.299,
          (float)-.308,(float)-.294,(float)-.276,(float)-.222,(float)-
.179,(
          float)- 062,(float)-
.011,(float)0.,(float).019,(float).09,(float)
          .208,(float).332,(float).361,(float).363,(float)1.2,(float)-
.283,(
          float)-.292,(float)-.281,(float)-.259,(float)-.223,(float)-
.217,(
          float)-.143,(float)-.07,(float)-
.012,(float)0.,(float).001,(float)

.121,(float).296,(float).424,(float).476,(float).489,(float)1.5,(
          float)-.262,(float)-.242,(float)-.236,(float)-.237,(float)-
.197,(
          float)-.176,(float)-.149,(float)-.056,(float)-.006,(float)0.,(

float).008,(float).099,(float).261,(float).381,(float).41,(float)
          .451,(float)2.,(float)-.205,(float)-.189,(float)-.225,(float)
          -.221,(float)-.235,(float)-.229,(float)-.217,(float)-
.191,(float)
          -.136,(float)-
.104,(float).069,(float).113,(float).224,(float).34,
          (float).374,(float).387,(float)4.,(float)-.156,(float)-.156,(
          float)-.156,(float)-.18,(float)-.191,(float)-.267,(float)-
.329,(
          float)-.431,(float)-.474,(float)-
.457,(float).362,(float).017,(

float).191,(float).319,(float).337,(float).355,(float)6.,(float)
          -.108,(float)-.116,(float)-.117,(float)-.169,(float)-
.2,(float)
          -.233,(float)-.297,(float)-.408,(float)-.43,(float)-
.429,(float)

.326,(float).022,(float).197,(float).309,(float).326,(float).343,(
          float)20.,(float)-.106,(float)-.113,(float)-.127,(float)-
.163,(
          float)-.192,(float)-.241,(float)-.305,(float)-.41,(float)-
.45,(
          float)-
.435,(float).361,(float).012,(float).17,(float).292,(float)
          .341,(float).338 };
    static integer icam1 = 0;

    static real mach, t, alfat;
    extern /* Subroutine */ int receive_net_32__();
    static real tstep, ca;
    extern /* Subroutine */ int tlu2ei_(), send_net_32__();

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_net_32__(&mach);
    receive_net_32__(&alfat);
    if (t > tstg1) {
/* second stage */
      tlu2ei_(&mach, &alfat, ca2m, &icam2, &icaa2, &ca);
    } else {
/* first stage */
      tlu2ei_(&mach, &alfat, ca1m, &icam1, &icaa1, &ca);
    }
```

```
    send_net_32__(&ca);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.2 Aerocn.c

```
/* aerocn.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static integer icna1 = 0;
    static integer icnm2 = 0;
    static integer icna2 = 0;
    static real tfinal = (float)62.501;
    static real tstg1 = (float)23.;
    static real cna1[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)

160.,(float)170.,(float)180.,(float)0.,(float)0.,(float).1943,(

float).4861,(float).6991,(float)1.031,(float)1.8471,(float)2.8683,
          (float)5.198,(float)6.7229,(float)6.9385,(float)6.7289,(float)

5.1903,(float)2.8804,(float).8334,(float).2054,(float)0.,(float)

.5,(float)0.,(float).2016,(float).4596,(float).7148,(float)1.0404,
          (float)1.8189,(float)2.877,(float)5.1897,(float)6.7274,(float)

6.9327,(float)6.7164,(float)5.2139,(float)2.8484,(float).7949,(
          float).2014,(float)0.,(float).8,(float)0.,(float).2108,(float)
          .4888,(float).7689,(float)1.1237,(float)1.9773,(float)3.0839,(
          float)5.6104,(float)7.2627,(float)7.4829,(float)7.2585,(float)

5.6183,(float)3.0957,(float).8721,(float).2244,(float)0.,(float)

.9,(float)0.,(float).2453,(float).5632,(float).8506,(float)1.2693,

(float)2.1842,(float)3.3373,(float)6.0308,(float)7.7796,(float)

8.0298,(float)7.7874,(float)6.0173,(float)3.3226,(float).9462,(
          float).2334,(float)0.,(float)1.1,(float)0.,(float).259,(float)
          .6233,(float).9733,(float)1.4721,(float)2.5275,(float)3.9615,(
          float)7.1644,(float)9.2488,(float)9.5353,(float)9.2653,(float)

7.1618,(float)3.9405,(float)1.1188,(float).2778,(float)0.,(float)

1.2,(float)0.,(float).249,(float).629,(float).9901,(float)1.4938,(
          float)2.7177,(float)4.2454,(float)7.7199,(float)9.9831,(float)

10.2965,(float)10.0024,(float)7.7086,(float)4.2635,(float)1.2194,(

float).3056,(float)0.,(float)1.5,(float)0.,(float).2727,(float)
          .625,(float)1.0606,(float)1.6051,(float)2.7252,(float)4.2279,(

float)7.6637,(float)9.9322,(float)10.2274,(float)9.9171,(float)
```

```
7.6573,(float)4.2184,(float)1.1961,(float).3133,(float)0.,(float)

2.,(float)0.,(float).2753,(float).6384,(float)1.1354,(float)1.664,

(float)2.7542,(float)3.9459,(float)7.1899,(float)9.2966,(float)

9.5868,(float)9.2911,(float)7.2048,(float)3.9474,(float)1.129,(
          float).2811,(float)0.,(float)4.,(float)0.,(float).2656,(float)
          .658,(float)1.189,(float)1.7299,(float)2.6718,(float)3.7776,(
          float)6.8236,(float)8.8635,(float)9.1188,(float)8.8535,(float)

6.8525,(float)3.7561,(float)1.0696,(float).2716,(float)0.,(float)
          6.,(float)0.,(float).2616,(float).6025,(float)1.0334,(float)

 .4472,(float)2.522,(float)3.7044,(float)6.7629,(float)8.7381,(
          float)9.0126,(float)8.7377,(float)6.7429,(float)3.7153,(float)

1.0442,(float).2684,(float)0.,(float)9.,(float)0.,(float).2567,(
          float).615,(float)1.0322,(float)1.4501,(float)2.5213,(float)

3.7166,(float)6.7689,(float)8.7283,(float)9.0069,(float)8.7303,(
          float)6.7754,(float)3.738,(float)1.0652,(float).2765,(float)0.
};
    static real cna2[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)

160.,(float)170.,(float)180.,(float)0.,(float)0.,(float).1526,(

float).3299,(float).5186,(float).7355,(float)1.3117,(float)1.7864,

(float)2.6227,(float)3.3848,(float)3.5119,(float)3.3989,(float)

2.6088,(float)1.4323,(float).4123,(float).105,(float)0.,(float).5,
          (float)0.,(float).1466,(float).324,(float).5095,(float).7414,(
          float)1.3098,(float)1.8016,(float)2.6256,(float)3.3828,(float)
          3.5111,(float)3.3918,(float)2.6245,(float)1.437,(float).4156,(
          float).1115,(float)0.,(float).8,(float)0.,(float).1389,(float)
          .3278,(float).5399,(float).7499,(float)1.4611,(float)2.0041,(
          float)2.8421,(float)3.6682,(float)3.796,(float)3.6815,(float)

2.8486,(float)1.5607,(float).465,(float).1158,(float)0.,(float).9,

(float)0.,(float).1547,(float).3536,(float).5956,(float).8675,(
          float)1.6577,(float)2.1614,(float)3.0455,(float)3.9357,(float)
          4.0622,(float)3.9195,(float)3.0225,(float)1.6694,(float).463,(

float).1169,(float)0.,(float)1.1,(float)0.,(float).1878,(float)
          .4249,(float).6982,(float)1.001,(float)1.6649,(float)2.3611,(
          float)3.619,(float)4.6821,(float)4.8318,(float)4.6926,(float)

3.6108,(float)1.4848,(float).56,(float).1396,(float)0.,(float)1.2,

(float)0.,(float).1929,(float).4189,(float).6926,(float)1.0156,(
          float)1.7242,(float)2.5269,(float)3.948,(float)5.0947,(float)
          5.254,(float)5.103,(float)3.9517,(float)2.1728,(float).6054,(

float).1699,(float)0.,(float)1.5,(float)0.,(float).2014,(float)

.412,(float).748,(float)1.0899,(float)1.7873,(float)2.5183,(float)
```

```
3.8838,(float)5.0173,(float)5.1797,(float)5.0229,(float)3.8697,(

float)2.1273,(float).6126,(float).173,(float)0.,(float)2.,(float)

0.,(float).1915,(float).4388,(float).7244,(float)1.0354,(float)
          1.751,(float)2.4106,(float)3.6931,(float)4.774,(float)4.9262,(
          float)4.7731,(float)3.7039,(float)2.0259,(float).5808,(float)

.1458,(float)0.,(float)4.,(float)0.,(float).2038,(float).4157,(
          float).6921,(float)1.0066,(float)1.6656,(float)2.2891,(float)

3.4786,(float)4.4804,(float)4.6223,(float)4.481,(float)3.4656,(

float)1.8978,(float).5393,(float).1339,(float)0.,(float)6.,(float)
          0.,(float).1866,(float).3974,(float).6419,(float).9172,(float)

1.4828,(float)2.1442,(float)3.4153,(float)4.4316,(float)4.552,(
          float)4.4333,(float)3.4277,(float)1.8834,(float).5306,(float)

.1412,(float)0.,(float)20.,(float)0.,(float).1656,(float).3955,(
          float).6324,(float).8981,(float)1.4796,(float)2.1259,(float)

3.4177,(float)4.4451,(float)4.5763,(float)4.4204,(float)3.4285,(
          float)1.8721,(float).5357,(float).149,(float)0. };
    static integer icnm1 = 0;

    static real mach, t, alfat, tstep;
    extern /* Subroutine */ int receive_real_32bit__(), tlu2ei_();
    static real cn;
    extern /* Subroutine */ int send_real_32bit__();

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&mach);
    receive_real_32bit__(&alfat);
    if (t > tstg1) {
/* second stage */
      tlu2ei_(&mach, &alfat, cna2, &icnm2, &icna2, &cn);
    } else {
/* first stage */
      tlu2ei_(&mach, &alfat, cna1, &icnm1, &icna1, &cn);
    }
    send_real_32bit__(&cn);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.3 Aeroxcp.c

```
/* aeroxcp.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static integer ixcpa1 = 0;
    static integer ixcpm2 = 0;
    static integer ixcpa2 = 0;
    static real tfinal = (float)62.501;
    static real tstg1 = (float)23.;
    static real xcpl1[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)

160.,(float)170.,(float)180.,(float)0.,(float)40.2,(float)55.6,(

float)59.4,(float)58.9,(float)59.5,(float)59.3,(float)62.3,(float)
          67.1,(float)78.8,(float)85.2,(float)104.8,(float)112.6,(float)
          116.8,(float)134.4,(float)155.8,(float)158.4,(float).5,(float)

42.1,(float)55.,(float)61.,(float)57.9,(float)61.2,(float)61.4,(
          float)61.2,(float)67.8,(float)78.5,(float)86.1,(float)106.5,(

float)115.1,(float)118.3,(float)132.2,(float)156.5,  .oat)157.8,(

float).8,(float)41.1,(float)59.9,(float)59.4,(float)63.,(float)

62.,(float)63.4,(float)63.5,(float)71.8,(float)78.2,(float)88.,(

float)114.1,(float)125.4,(float)124.,(float)137.2,(float)159.8,(

float)161.1,(float).9,(float)40.3,(float)56.8,(float)62.2,(float)

63.8,(float)62.1,(float)64.4,(float)65.3,(float)72.2,(float)78.3,(

float)84.8,(float)117.8,(float)125.8,(float)127.7,(float)140.4,(
          float)166.9,(float)175.9,(float)1.1,(float)42.1,(float)58.6,(

float)60.3,(float)63.,(float)63.5,(float)63.8,(float)64.5,(float)
          69.,(float)79.9,(float)85.7,(float)123.4,(float)130.3,(float)
          130.9,(float)147.6,(float)188.,(float)197.5,(float)1.2,(float)

43.5,(float)55.8,(float)60.3,(float)63.6,(f' .at)63.2,(float)63.3,(

float)68.3,(float)74.2,(float)80.6,(float)86.,(float)127.3,(float)

131.4,(float)134.,(float)152.3,(float)189.7,(float)198.3,(float)

1.5,(float)43.5,(float)57.6,(float)59.5,(floac)62.3,(float)61.4,(

float)66.9,(float)72.5,(float)76.9,(float)82.8,(float)85.7,(float)
```

```
95.2,(float)101.9,(float)105.4,(float)111.5,(float)147.6,(float)
160.2,(float)2.,(float)46.,(float)59.3,(float)63.7,(float)65.4,(
float)67.,(float)73.3,(float)75.4,(float)81.2,(float)82.6,(float)
86.,(float)101.6,(float)104.8,(float)107.8,(float)109.7,(float)
129.6,(float)145.7,(float)4.,(float)51.3,(float)60.2,(float)68.3,(
float)69.9,(float)72.1,(float)77.9,(float)79.9,(float)80.9,(float)
84.2,(float)86.6,(float)113.9,(float)116.3,(float)119.3,(float)
121.2,(float)125.9,(float)134.9,(float)6.,(float)52.3,(float)64.2,
(float)66.4,(float)69.,(float)68.,(float)74.3,(float)75.5,(float)
          81.9,(float)82.5,(float)86.1,(float)117.2,(float)120.7,(float)
124.,(float)126.6,(float)128.8,(float)134.4,(float)9.,(float)50.8,
          (float)62.6,(float)66.6,(float)70.1,(float)68.5,(float)73.2,(
          float)76.6,(float)81.3,(float)83.3,(float)86.4,(float)117.3,(
          float)122.6,(float)125.2,(float)126.,(float)129.8,(float)135.2
};
    static real xcp12[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(
float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(
float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)
160.,(float)170.,(float)180.,(float)0.,(float)33.5,(float)35.,(
float)35.3,(float)36.4,(float)36.8,(float)38.,(float)37.9,(float)
42.4,(float)45.2,(float)45.6,(float)52.6,(float)59.,(float)59.6,(
float)69.4,(float)79.8,(float)82.9,(float).5,(float)35.2,(float)
33.4,(float)33.3,(float)36.4,(float)35.5,(float)38.4,(float)40.4,(
float)42.9,(float)45.4,(float)47.7,(float)51.4,(float)57.2,(float)
59.3,(float)68.1,(float)80.7,(float)80.1,(float).8,(float)34.4,(
float)34.8,(float)35.1,(float)36.6,(float)38.5,(float)40.3,(float)
41.4,(float)43.5,(float)46.6,(float)47.6,(float)53.5,(float)63.1,(
float)63.5,(float)71.1,(float)80.8,(float)83.7,(float).9,(float)
35.7,(float)36.,(float)35.1,(float)35.8,(float)38.2,(float)40.9,(
float)42.8,(float)44.3,(float)46.7,(float)45.8,(float)53.6,(float)
65.1,(float)67.,(float)71.7,(float)86.8,(float)90.6,(float)1.1,(
float)37.9,(float)38.6,(float)36.4,(float)37.4,(float)38.4,(float)
41.2,(float)43.6,(float)42.8,(float)43.6,(float)47.9,(float)56.7,(
float)67.4,(float)67.6,(float)77.,(float)96.2,(float)102.6,(float)
```

```
1.2,(float)36.7,(float)35.1,(float)36.5,(float)35.,(float)40.,(
float)41.7,(float)41.1,(float)43.,(float)46.7,(float)47.7,(float)
59.2,(float)66.5,(float)67.6,(float)80.,(float)97.9,(float)102.7,(
float)1.5,(float)34.7,(float)35.2,(float)36.7,(float)39.5,(float)
40.5,(float)40.9,(float)41.7,(float)45.,(float)45.7,(float)47.5,(
float)49.5,(float)53.5,(float)55.5,(float)59.3,(float)75.8,(float)
        82.4,(float)2.,(float)35.,(float)36.1,(float)37.,(float)38.4,(
float)40.5,(float)41.,(float)41.,(float)44.4,(float)45.5,(float)
47.6,(float)52.6,(float)54.3,(float)55.6,(float)58.,(float)67.3,(
float)76.5,(float)4.,(float)33.7,(float)36.4,(float)37.3,(float)
39.7,(float)41.,(float)43.6,(float)41.5,(float)44.4,(float)45.7,(
float)46.4,(float)53.8,(float)60.6,(float)60.8,(float)62.8,(float)
65.2,(float)70.,(float)6.,(float)37.8,(float)36.2,(float)37.6,(
float)39.9,(float)39.6,(float)42.2,(float)43.4,(float)43.,(float)
46.9,(float)48.7,(float)54.3,(float)62.4,(float)65.1,(float)64.3,(
float)68.2,(float)70.5,(float)20.,(float)35.,(float)36.1,(float)
37.9,(float)39.5,(float)40.3,(float)40.7,(float)42.,(float)42.5,(
float)46.8,(float)48.7,(float)52.8,(float)62.,(float)64.5,(float)
        65.1,(float)67.2,(float)68.8 };
    static integer ixcpm1 = 0;

    static real mach, t, alfat, tstep;
    extern /* Subroutine */ int receive_real_32bit__(), tlu2ei_(),
        send_real_32bit__();
    static real xcp;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&mach);
    receive_real_32bit__(&alfat);
    if (t > tstg1) {
/* second stage */
      tlu2ei_(&mach, &alfat, xcpl2, &ixcpm2, &ixcpa2, &xcp);
    } else {
/* first stage */
      tlu2ei_(&mach, &alfat, xcpl1, &ixcpm1, &ixcpa1, &xcp);
    }
    xcp = -(doublereal)xcp / (float)12.;
    send_real_32bit__(&xcp);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
```

```
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.4 Attlm.c

```
/* attlm.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__5 = 5;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real attltt[5] = {
(float)0.,(float)7.5,(float)11.,(float)19.25,(
          float)100. };
    static real attlmt[5] = {
(float).015,(float).015,(float).093,(float).41,(
          float).41 };
    static integer itable = 0;

    static real t;
    extern /* Subroutine */ int table_();
    static real attlm, tstep;
    extern /* Subroutine */ int send_real_32bit__();

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    table_(attltt, attlmt, &t, &attlm, &c__5, &itable);
    send_real_32bit__(&attlm);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.5 Bauto.c

```
/* bauto.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__59 = 59;
static integer c__29 = 29;
static integer c__26 = 26;
static doublereal c_b7 = 1.4;
static integer c__4 = 4;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real presste[59] = {
(float)2116.25,(float)1967.72,(float)1827.78,(

float)1696.02,(float)1571.91,(float)1455.35,(float)1345.9,(float)

1243.2,(float)1147.72,(float)1057.49,(float)973.289,(float)857.26,
          (float)752.725,(float)658.783,(float)574.592,(float)499.356,(

float)432.644,(float)374.755,(float)324.623,(float)281.21,(float)
          243.614,(float)203.13,(float)166.178,(float)137.264,(float)
          113.389,(float)93.7284,(float)77.5639,(float)64.2586,(float)
          53.2944,(float)44.2494,(float)37.1196,(float)29.2323,(float)
          23.2726,(float)18.5578,(float)14.8375,(float)11.912,(float)
          9.60154,(float)7.76921,(float)6.30961,(float)5.14276,(float)
          4.20625,(float)3.45183,(float)2.84192,(float)2.34714,(float)
          1.94196,(float)1.60687,(float)1.32973,(float)1.10007,(float)
          .908378,(float).61551,(float).413455,(float).274355,(float)
          .178439,(float).113488,(float).070406,(float).042482,(float)
          .017169,(float).002643,(float).002643 };
    static real vsndte[59] = {
(float)1116.45,(float)1108.75,(float)1100.99,(

float)1093.19,(float)1085.32,(float)1077.4,(float)1069.43,(float)
          1061.39,(float)1053.3,(float)1045.15,(float)1036.93,(float)

1024.48,(float)1011.89,(float)999.14,(float)986.22,(float)973.14,(
          float)968.08,(float)968.08,(float)968.08,(float)968.08,(float)

968.08,(float)968.08,(float)968.08,(float)968.08,(float)968.19,(
          float)970.9,(float)973.59,(float)976.14,(float)978.95,(float)

987.62,(float)984.28,(float)987.59,(float)990.9,(float)994.18,(

float)1002.72,(float)1011.79,(float)1020.77,(float)1029.67,(float)
          1038.48,(float)1047.22,(float)1055.88,(float)1064.47,(float)
          1072.99,(float)1081.43,(float)1082.02,(float)1082.02,(float)
          1082.02,(float)1078.43,(float)1072.4,(float)1060.25,(float)

1047.98,(float)1025.62,(float)1000.11,(float)973.96,(float)947.12,
          (float)919.5,(float)884.,(float)894.5,(float)894.5 };
```

```
    static real timte1[26] = {
(float)0.,(float).01,(float).02,(float).04,(

float).06,(float).08,(float).1,(float).14,(float).2,(float)1.,(

float)2.,(float)5.5,(float)7.,(float)12.,(float)16.,(float)18.,(

float)20.,(float)22.,(float)23.,(float)23.5,(float)23.7,(float)
          23.8,(float)24.,(float)24.4,(float)24.551,(float)24.552 };
    static real timte2[29] = {
(float)0.,(float).01,(float).02,(float).04,(

float).06,(float).08,(float).1,(float).14,(float).2,(float)1.,(

float)3.,(float)5.,(float)6.5,(float)8.,(float)10.,(float)12.,(
          float)15.,(float)19.5,(float)27.,(float)31.,(float)33.,(float)

36.5,(float)37.1,(float)37.2,(float)37.5,(float)37.6,(float)37.8,(
          float)38.254,(float)38.255 };
    static real thrte1[26] = {
(float)0.,(float)58.032,(float)598.508,(float)

2645.385,(float)4000.256,(float)4622.207,(float)4939.192,(float)

5244.599,(float)5517.307,(float)6016.711,(float)7063.661,(float)

7903.243,(float)7989.524,(float)9142.214,(float)9607.177,(float)

9737.093,(float)9393.564,(float)8788.951,(float)8267.048,(float)

7558.824,(float)3181.193,(float)1523.628,(float)363.282,(float)
          18.646,(float)6.197,(float)0. };
    static real thrte2[29] = {
(float)0.,(float)55.723,(float)551.693,(float)

2156.503,(float)2879.546,(float)3044.879,(float)3005.564,(float)

2889.39,(float)2860.064,(float)2880.036,(float)3319.566,(float)

3577.371,(float)3532.261,(float)3543.431,(float)3589.937,(float)

3641.641,(float)3864.178,(float)4045.921,(float)3960.807,(float)

3880.334,(float)3843.856,(float)3599.69,(float)3395.138,(float)

2969.519,(float)523.301,(float)265.601,(float)70.724,(float)6.019,
          (float)0. };
    static real tapu = (float)0.;
    static real dtapu = (float).005;
    static real tapustep = (float)5.;
    static real mchlim = (float)4.;
    static real tfinal = (float)62.501;
    static real kthtk1 = (float).6;
    static real kthtk2 = (float)1.5;
    static real tign = (float).01;
    static real tst2on = (float)22.995;
    static real tfrcs = (float)23.;
    static real tmode2 = (float)23.01;
    static real wmrvc = (float)25.;
    static real zettvc = (float).85;
    static real wmfrtt[4] = {
(float)0.,(float)9.5,(float)39.95,(float)100. };
```

```
    static real wmfrct[4] = {
(float)62.83,(float)62.83,(float)42.,(float)42.
          };
    static real tstg1 = (float)23.;
    static real zetfrc = (float).85;
    static real delon = (float).045;
    static real bcklmt = (float).15;
    static real delthg = (float).045;
    static real thjet = (float)370.;
    static real sjet = (float)1.3273;
    static real sref1 = (float)1.968953;
    static real sref2 = (float)1.968953;
    static real aexite = (float).305;
    static real aexit2 = (float).99;
    static real tstg2 = (float)62.5;
    static real xnoze = (float)-12.5583;
    static real xnoz2 = (float)-7.39167;
    static real djet = (float)1.3;
    static real xjet = (float)-2.71;
    static integer ialte = 0;
    static integer ith1e = 0;
    static integer ith2e = 0;
    static integer iwmfrc = 0;
    static real dtr = (float).017453292519943296;
    static real slglbm = (float)32.174048;
    static real dteps = (float)1e-13;
    static real frcloc[12]    /* was [3][4] */ = { (float)-
2.628,(float).73,
          (float)0.,(float)-2.628,(float)0.,(float).73,(float)-
2.628,(float)
          -.73,(float)0.,(float)-2.628,(float)0.,(float)-.73 };
    static real altte[59] = {
(float)0.,(float)2e3,(float)4e3,(float)6e3,(
          float)8e3,(float)1e4,(float)1.2e4,(float)1.4e4,(float)1.6e4,(

float)1.8e4,(float)2e4,(float)2.3e4,(float)2.6e4,(float)2.9e4,(

float)3.2e4,(float)3.5e4,(float)3.8e4,(float)4.1e4,(float)4.4e4,(

float)4.7e4,(float)5e4,(float)5.4e4,(float)5.8e4,(float)6.2e4,(

float)6.6e4,(float)7e4,(float)7.4e4,(float)7.8e4,(float)8.2e4,(
          float)8.6e4,(float)9e4,(float)9.5e4,(float)1e5,(float)1.05e5,(

float)1.1e5,(float)1.15e5,(float)1.2e5,(float)1.25e5,(float)1.3e5,
          (float)1.35e5,(float)1.4e5,(float)1.45e5,(float)1.5e5,(float)
          1.55e5,(float)1.6e5,(float)1.65e5,(float)1.7e5,(float)1.75e5,(

float)1.8e5,(float)1.9e5,(float)2e5,(float)2.1e5,(float)2.2e5,(

float)2.3e5,(float)2.4e5,(float)2.5e5,(float)2.75e5,(float)3e5,(
          float)999999. };
    static real rhote[59] = {
(float)76474.,(float)72098.,(float)67917.,(
          float)63925.,(float)60116.,(float)56483.,(float)53022.,(float)

49725.,(float)46589.,(float)43606.,(float)40773.,(float)36790.,(
          float)33113.,(float)29725.,(float)26610.,(float)23751.,(float)

20794.,(float)18012.,(float)15602.,(float)13516.,(float)11709.,(
          float)9670.1,(float)7987.,(float)6597.3,(float)5448.5,(float)

4478.7,(float)3685.8,(float)3036.8,(float)2504.9,(float)2068 5,(
```

```
float)1710.,(float)1350.,(float)1067.6,(float)845.7,(float)664.7,(
          float)524.12,(float)415.06,(float)330.06,(float)263.53,(float)
          211.22,(float)169.94,(float)137.22,(float)111.19,(float)90.4,(
          float)74.713,(float)61.822,(float)51.159,(float)42.605,(float)

35.578,(float)24.663,(float)16.957,(float)11.748,(float)8.0356,(

float)5.3888,(float)3.5353,(float)2.263,(float).6171,(float).1488,
          (float).1488 };

    /* System generated locals */
    real r_1, r_2, r_3, r_4;
    doublereal d_1;

    /* Local variables */
    static real cmmd[2], dlpc, dlyc, xdel, xcpe, thre, kpsi, ktht;
    extern doublereal sparctan_();
    static real vrwm[3], t;
    extern /* Subroutine */ int table_();
    static real cnalp, srefe, cgest[3], xcpcg, kpsid, kthtd, estqa,
psier,
          thter, wmfrc, thrve, tstep, t0;
    extern /* Subroutine */ int receive_real_32bit__();
    static real kthfm1, kthfm2, ld, ii[3], ct, alfate, malpha, sq, sr,
lfracs,
           kthfrc, estmch, mdltfr, totcmd, krtfrc, estalt;
    extern /* Subroutine */ int send_real_32bit__();
    static real estrho, estpre, estvel, estvsd, cne, kme, kne, alt;

/*
-----------------------------------------------------------------
------ */
/*      subroutine bauto(t,thter,psier,sq,sr,ii,cgest,vrwm,alt,cmmd, */
/*     .                 dlpc,dlyc,mdltfr,malpha) */
/*
-----------------------------------------------------------------
------ */
/*     function :            provides control of the missile about three
*/
/*                           axes throughout the boost phase of flight
*/
/*     inputs :              t,thter,psier,sq,sr,ii,cgest,vrwm,alt */
/*     outputs :             cmmd,dlpc,dlyc,mdltfr,malpha */
/*
-----------------------------------------------------------------
------ */
/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    cmmd[0] = (float)0.;
    cmmd[1] = (float)0.;
    dlpc = (float)0.;
    dlyc = (float)0.;
    mdltfr = (float)0.;
    malpha = (float)0.;
    sq = (float)0.;
    sr = (float)0.;
L10:
    send_real_32bit__(cmmd);
    send_real_32bit__(&cmmd[1]);
    send_real_32bit__(&dlpc);
    send_real_32bit__(&dlyc);
    send_real_32bit__(&sq);
```

```
    send_real_32bit__(&sr);
    send_real_32bit__(&mdltfr);
    send_real_32bit__(&malpha);
    receive_real_32bit__(cgest);
    receive_real_32bit__(&cgest[1]);
    receive_real_32bit__(&cgest[2]);
    receive_real_32bit__(&ii[1]);
    receive_real_32bit__(&alt);
    receive_real_32bit__(vrwm);
    receive_real_32bit__(&vrwm[1]);
    receive_real_32bit__(&vrwm[2]);
    if (tstep >= tapu) {
      tapu += tapustep;
      if (t < tstg2) {
          if ((r_1 = t - tstg1, dabs(r_1)) <= dteps) {
            aexite = aexit2;
            xnoze = xnoz2;
          }
          estalt = alt;
          table_(altte, rhote, &estalt, &estrho, &c__59, &ialte);
          estrho = estrho * (float)1e-6 / slglbm;
          table_(altte, prestte, &estalt, &estpre, &c__59, &ialte);
          table_(altte, vsndte, &estalt, &estvsd, &c__59, &ialte);
/* Computing 2nd power */
          r_1 = vrwm[0];
/* Computing 2nd power */
          r_2 = vrwm[1];
/* Computing 2nd power */
          r_3 = vrwm[2];
          estvel = sqrt(r_1 * r_1 + r_2 * r_2 + r_3 * r_3);
          estmch = estvel / estvsd;
/* Computing 2nd power */
          r_1 = estvel;
          estqa = estrho * (r_1 * r_1) / (float)2.;
          if (t > tstg1) {
            srefe = sref2;
            t0 = t - tst2on;
            table_(timte2, thrte2, &t0, &thrve, &c__29, &ith2e);
          } else {
            t0 = t - tign;
            srefe = sref1;
            table_(timte1, thrte1, &t0, &thrve, &c__26, &ith1e);
          }
          thre = thrve - aexite * estpre;
          if (thre < (float)0.) {
            thre = (float)0.;
          }
          if (estvel > (float)0.) {
/* Computing 2nd power */
            r_2 = vrwm[1];
/* Computing 2nd power */
            r_3 = vrwm[2];
            r_1 = sqrt(r_2 * r_2 + r_3 * r_3);
            r_4 = dabs(vrwm[0]);
            alfate = sparctan_(&r_1, &r_4) / dtr;
          } else {
            alfate = (float)0.;
          }
          send_real_32bit__(&estmch);
          send_real_32bit__(&alfate);
/*          if ( t.lt.tstg1 ) then */
/*             call tlu2ei(estmch,4.0d0,cnale,icnmle,icnale,cne)
*/
/*             call tlu2ei(estmch,alfate,xcplle,icpmle,icpale,
```

```
xcpe) */
/*           else */
/*              call tlu2ei(estmch,4.0d0,cna2e,icnm2e,icna2e,cne)
*/
/*              call tlu2ei(estmch,alfate,xcpl2e,icpm2e,icpa2e,
xcpe) */
/*           end if */
          receive_real_32bit__(&cne);
          receive_real_32bit__(&xcpe);
/*  conversion from inches to feet */
          xcpe = -(doublereal)xcpe / (float)12.;
/*  calculate cnalfa (per radian) */
          cnalp = cne / (dtr * (float)4.);
          xcpcg = xcpe - cgest[0];
          if (thre >= (float)1e3 && ii[1] > (flcat)1e-6) {
            malpha = (r_1 = cnalp * xcpcg * srefe * estqa / ii[1], dabs(
                r_1));
          }
          receive_real_32bit__(&psier);
          receive_real_32bit__(&thter);
          receive_real_32bit__(&sq);
          receive_real_32bit__(&sr);
/*  tvc autopilot */
          if (t < tmode2) {
            if (thre >= (float)1e3 && ii[1] > (float)1e-6) {
              xdel = cgest[0] - xnoze;
/* Computing 2nd power */
              r_1 = wmtvc;
              ktht = (ii[1] * (r_1 * r_1) + cnalp * srefe * estqa *
                    xcpcg) / (thre * xdel);
/* Computing 2nd power */
              r_1 = wmtvc;
              kpsi = (ii[1] * (r_1 * r_1) + cnalp * srefe * estqa *
                    xcpcg) / (thre * xdel);
              kthtd = zettvc * (float)2. * wmtvc * ii[1] / (thre *
xdel)
                    ;
              kpsid = zettvc * (float)2. * wmtvc * ii[1] / (thre *
xdel)
                    ;
            } else {
              ktht = (float)4.;
              kpsi = (float)4.;
              kthtd = (float)4.;
              kpsid = (float)4.;
            }
            cmmd[0] = thter * ktht - sq * kthtd;
            cmmd[1] = psier * kpsi - sr * kpsid;
/* Computing 2nd power */
            r_1 = cmmd[0];
/* Computing 2nd power */
            r_2 = cmmd[1];
            totcmd = sqrt(r_1 * r_1 + r_2 * r_2);
            if (totcmd > bcklmt) {
              cmmd[0] = cmmd[0] * bcklmt / totcmd;
              cmmd[1] = cmmd[1] * bcklmt / totcmd;
            }
          } else {
            cmmd[0] = (float)0.;
            cmmd[1] = (float)0.;
          }
/*  forward reaction control system autopilot */
          if (t >= tfrcs) {
            if (thre >= (float)1e3 && ii[1] > (float)1e-6) {
```

```
                ld = (xjet - xnoze) / djet;
                ct = thjet / (estqa * sjet);
                if (estmch <= mchlim) {
                  kne = ((float)1. - sqrt(ld) * (float).485) * (float)
                        .1358 / sqrt(ct) + (float).6118 + estmch * (
                        float).0946 + (float).004317 / ld;
                } else {
                  d_1 = (doublereal) (log(ct) + (float)8.5);
                  r_1 = (float)1.1 - pow_dd(&d_1, &c_b7) * (float).2116;

                  kne = exp(r_1) + (float)1.;
                }
                kme = (float).5582 - (float).1884 / sqrt(ct) - (float)
                      1.9659 / ld;
                lfracs = frcloc[0] - cgest[0];
                mdltfr = (-(doublereal)kme * thjet * djet + kne * thjet
*
                      lfracs) / ii[1];
                r_1 = t - tmode2;
                table_(wmfrtt, wmfrct, &r_1, &wmfrc, &c__4, &iwmfrc);
/* Computing 2nd power */
                r_1 = wmfrc;
                krtfrc = zetfrc * (float)2. * wmfrc / (r_1 * r_1 +
malpha)
                      ;
                kthfrc = delon * (float)2. / (mdltfr * krtfrc * dtapu);
                kthfm1 = delon * malpha / mdltfr;
                kthfm2 = delon / delthg;
                if (kthfrc < kthfm1) {
                  kthfrc = kthfm1;
                }
                if (kthfrc < kthfm2) {
                  kthfrc = kthfm2;
                }
                ktht = kthfrc * kthtk1;
                kthtd = ktht * krtfrc * kthtk2;
                kpsi = ktht;
                kpsid = kthtd;
              } else {
                malpha = (float)544.18;
                mdltfr = (float)6.0437;
                ktht = (float)10.;
                kthtd = (float)25.;
                kpsi = (float)10.;
                kpsid = (float)25.;
              }
              dlpc = thter * ktht - sq * kthtd;
              dlyc = psier * kpsi - sr * kpsid;
            }
          } else {
            send_real_32bit__(&estmch);
            send_real_32bit__(&alfate);
            receive_real_32bit__(&cne);
            receive_real_32bit__(&xcpe);
            receive_real_32bit__(&psier);
            receive_real_32bit__(&thter);
            receive_real_32bit__(&sq);
            receive_real_32bit__(&sr);
          }
        } else {
          send_real_32bit__(&estmch);
          send_real_32bit__(&alfate);
          receive_real_32bit__(&cne);
          receive_real_32bit__(&xcpe);
```

```
      receive_real_32bit__(&psier);
      receive_real_32bit__(&thter);
      receive_real_32bit__(&sq);
      receive_real_32bit__(&sr);
    }
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.6 Boost2a.c

```
/* boost2a.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__1 = 1;
static integer c__2 = 2;
static integer c__3 = 3;
static integer c__5 = 5;
static integer c__6 = 6;
static integer c__7 = 7;
static integer c__8 = 8;
static integer c__9 = 9;
static integer c__10 = 10;
static integer c__11 = 11;
static real c_b15 = (float)0.;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static doublereal delt = .001;
    static doublereal slglbm = 32.174048;
    static real mass0 = (float)43.939;
    static real wkv0 = (float)97.1;
    static doublereal latlp = 0.;
    static doublereal longlp = 0.;
    static doublereal tfinal = 62.501;
    static doublereal tstg1 = 23.;
    static doublereal tstg2 = 62.5;
    static doublereal dteps = 1e-13;
    static doublereal rade = 20898908.;
    static real msstg2 = (float)19.457;
    static doublereal dtr = .017453292519943296;

    /* System generated locals */
    doublereal d_1, d_2, d_3;

    /* Local variables */
    static real frcx, mass, frcy, frcz, mdot;
    static doublereal xyzd[3], xyze[3];
    extern /* Subroutine */ int spintegi_();
    static integer i;
    static doublereal t;
    extern /* Subroutine */ int integ_();
    static real mdotf, mdott;
    static doublereal tstep, xyzdd[3], xyzed[3];
    extern /* Subroutine */ int receive_real_32bit__();
    static real gr[3], ud, vd, wd;
    extern /* Subroutine */ int integi_(), send_real_32bit__(),
missil_(),
          vecrot_(), send_real_64bit__();
    static integer mdotkv;
    static doublereal xyzedd[3];
    static real wdotkv;
    static doublereal cei[9];
    static real cim[9], alt;
```

```
    extern /* Subroutine */ int mmk_();
    static real fxt, fyt, fzt, spt, wkv, massold;
    extern /* Subroutine */ int spinteg_();
    static doublereal xyz[3];
    static real tmp_xyz__[3];

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    for (i = 1; i <= 3; ++i) {
      xyze[i - 1] = (float)0.;
      xyzed[i - 1] = (float)0.;
      xyzedd[i - 1] = (float)0.;
/* L10: */
    }
    xyze[0] = rade;
/* Computing 2nd power */
    d_1 = xyze[0];
/* Computing 2nd power */
    d_2 = xyze[1];
/* Computing 2nd power */
    d_3 = xyze[2];
    alt = sqrt(d_1 * d_1 + d_2 * d_2 + d_3 * d_3) - rade;
/*
-------------------------------------------------------------------
-----c */
/* ---------------------------- missile state initialization module
----c */
/*
-------------------------------------------------------------------
-----c */
/*   initialize states and state derivatives */
    mass = mass0;
    wkv = wkv0;
    d_1 = dtr * (float)-90.;
    d_2 = latlp * dtr;
    d_3 = longlp * dtr;
    mmk_(&d_1, &c__1, &d_2, &c__2, &d_3, &c__3, cei);
    vecrot_(xyzed, cei, xyzd);
    vecrot_(xyze, cei, xyz);
    mdot = (float)0.;
    wdotkv = (float)0.;
    vecrot_(xyzedd, cei, xyzdd);
    spt = t;
    spintegi_(&mass, &mdot, &spt, &c__1);
    spintegi_(&wkv, &wdotkv, &spt, &c__5);
    integi_(xyzd, xyzdd, &t, &c__6);
    integi_(&xyzd[1], &xyzdd[1], &t, &c__7);
    integi_(&xyzd[2], &xyzdd[2], &t, &c__8);
    integi_(xyz, xyzd, &t, &c__9);
    integi_(&xyz[1], &xyzd[1], &t, &c__10);
    integi_(&xyz[2], &xyzd[2], &t, &c__11);
/*
-------------------------------------------------------------------
------ */
/*  initialize processor inputs if not already initialized */
/* p1 */
    fxt = (float)0.;
    fyt = (float)0.;
    fzt = (float)0.;
    mdott = (float)0.;
    frcx = (float)0.;
    frcy = (float)0.;
    frcz = (float)0.;
```

```
    mdotf = (float)0.;
/* p2 */
/*  p3 */
    ud = (float)0.;
    vd = (float)0.;
    wd = (float)0.;
/*  p4 */
    for (i = 1; i <= 3; ++i) {
      gr[i - 1] = (float)0.;
/* L20: */
    }
/*  p5 */
/*  initialization routine */
    mass += delt * mdot;
/*
-----------------------------------------------------------------
-----c */
/* ---------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L30:
/*
*****************************************************************************
 */
/*                                                              * */
/*                              partition 1                                * */
/*                                                              * */
/*
*****************************************************************************
 */
/* ---------------------------- missile state update module
--------------c */
/*  temporarily extrapolate missile states from last integration */
/*  step ( note : the extrapolated states are overwritten when */
/*                the true integration is performed ) */
/* ------------- send parameters to partitions 3, 4, and 5
--------------c */
    send_real_32bit__(&ud);
    send_real_32bit__(&vd);
    send_real_32bit__(&wd);
    send_real_32bit__(gr);
    send_real_32bit__(&gr[1]);
    send_real_32bit__(&gr[2]);
    send_real_32bit__(&alt);
/* ---------- send mass to masspr subroutine table lookup processors
----c */
    send_real_32bit__(&mass);
    xyzd[0] += delt * xyzdd[0];
    xyzd[1] += delt * xyzdd[1];
    xyzd[2] += delt * xyzdd[2];
    xyz[0] += delt * xyzd[0];
    xyz[1] += delt * xyzd[1];
    xyz[2] += delt * xyzd[2];
/*  calculate current missile altitude */
/* Computing 2nd power */
    d_1 = xyz[0];
/* Computing 2nd power */
    d_2 = xyz[1];
/* Computing 2nd power */
    d_3 = xyz[2];
    alt = sqrt(d_1 * d_1 + d_2 * d_2 + d_3 * d_3) - rade;
/* ---------- send altitude to atmos subroutine table lookup processors
```

```
-c */
    send_real_32bit__(&alt);
/* ---------- send parameters to thread containing winds subroutine
-----c */
    send_real_64bit__(xyz);
    send_real_64bit__(&xyz[1]);
    send_real_64bit__(&xyz[2]);
    send_real_64bit__(xyzd);
    send_real_64bit__(&xyzd[1]);
    send_real_64bit__(&xyzd[2]);
    receive_real_32bit__(cim);
    receive_real_32bit__(&cim[1]);
    receive_real_32bit__(&cim[2]);
    receive_real_32bit__(&cim[3]);
    receive_real_32bit__(&cim[4]);
    receive_real_32bit__(&cim[5]);
    receive_real_32bit__(&cim[6]);
    receive_real_32bit__(&cim[7]);
    receive_real_32bit__(&cim[8]);
/* ---------- receive parameters from partition #2
----------------------c */
    receive_real_32bit__(&fxt);
    receive_real_32bit__(&fyt);
    receive_real_32bit__(&fzt);
    receive_real_32bit__(&frcx);
    receive_real_32bit__(&frcy);
    receive_real_32bit__(&frcz);
    receive_real_32bit__(&mdott);
    receive_real_32bit__(&mdotf);
    wkv += delt * wdotkv;
    mdotkv = (integer) (-(doublereal)mdotf * slglbm);
    mdot = -(doublereal)mdott - mdotf;
/*  save mass value for use in missil subroutine */
    massold = mass;
    spt = t;
/*  trapezoidal integration for simplicity */
    if ((d_1 = t - tstg1, abs(d_1)) <= dteps) {
/*  first stage separation */
      mass = msstg2;
      spintegi_(&mass, &c_b15, &spt, &c__1);
    } else if ((d_1 = t - tstg2, abs(d_1)) <= dteps) {
/*  second stage separation */
      mass = wkv / slglbm;
      spintegi_(&mass, &c_b15, &spt, &c__1);
    } else {
      spinteg_(&mass, &mdot, &spt, &c__1);
    }
    wkv = dmax(wkv,(float)0.);
    spinteg_(&wkv, &wdotkv, &spt, &c__5);
    mass += delt * mdot;
/* ---------------------------- vehicle states module
------------------c */
    missil_(&t, &massold, &fxt, &frcx, &fyt, &frcy, &fzt, &frcz, xyz,
xyzd, &
            ud, &vd, &wd, gr, cim, xyzdd);
/*
-----------------------------------------------------------------
-----c */
/*                              missile state integration module
c
*/
/*
-----------------------------------------------------------------
-----c */
```

```
    integ_(xyzd, xyzdd, &t, &c__6);
    integ_(&xyzd[1], &xyzdd[1], &t, &c__7);
    integ_(&xyzd[2], &xyzdd[2], &t, &c__8);
    integ_(xyz, xyzd, &t, &c__9);
    integ_(&xyz[1], &xyzd[1], &t, &c__10);
    integ_(&xyz[2], &xyzd[2], &t, &c__11);
/*  calculate current missile altitude */
/* Computing 2nd power */
    d_1 = xyz[0];
/* Computing 2nd power */
    d_2 = xyz[1];
/* Computing 2nd power */
    d_3 = xyz[2];
    alt = sqrt(d_1 * d_1 + d_2 * d_2 + d_3 * d_3) - rade;
    tmp_xyz__[0] = xyz[0];
    tmp_xyz__[1] = xyz[1];
    tmp_xyz__[2] = xyz[2];
    send_real_32bit__(tmp_xyz__);
    send_real_32bit__(&tmp_xyz__[1]);
    send_real_32bit__(&tmp_xyz__[2]);
    send_real_32bit__(&alt);
/*
**********************************************************************
 */
/*                                                                  * */
/*                         end of partition 1
 * */
/*                                                                  * */
/*
**********************************************************************
 */
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L30;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.7 Boost2a1.c

```
/* boost2a1.f -- translated by f2c (version of 3 February 1990
3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__1 = 1;
static integer c__2 = 2;
static integer c__3 = 3;
static doublereal c_b6 = 0.;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static doublereal delt = .001;
    static doublereal sref2 = 1.968953;
    static doublereal latlp = 0.;
    static doublereal longlp = 0.;
    static doublereal tmp1 = 0.;
    static doublereal tfinal = 62.501;
    static doublereal tstg1 = 23.;
    static doublereal tstg2 = 62.5;
    static doublereal omegae = 0.;
    static doublereal dtr = .0174532925199.3296;
    static doublereal sref1 = 1.968953;

    /* System generated locals */
    real r_1, r_2, r_3;
    doublereal d_1, d_2, d_3;

    /* Local variables */
    static real mach, long_, vsnd;
    static doublereal vrwi[3];
    static real vrwm[3];
    static doublereal xyzd[3], xyze[3], xyzr[3];
    static real rhod2;
    static doublereal a, b, c, d;
    static integer i;
    static doublereal t, cphia;
    static real alfat, shear;
    static doublereal sphia;
    static real cwdir, vwind, swdir;
    extern /* Subroutine */ int trans_();
    static doublereal tstep;
    extern /* Subroutine */ int receive_real_32bit__(), mmlxy_();
    static doublereal mvrwm;
    extern /* Subroutine */ int receive_real_64bit__();
    static real ca, cn, qa;
    static doublereal qs;
    extern doublereal arctan_();
    extern /* Subroutine */ int vecsub_(), send_real_32bit__();
    static doublereal viwind[3];
    extern /* Subroutine */ int vecrot_();
    static doublereal vrwind[3], vwwind[3], cie[9], cei[9];
    static real cim[9];
    static doublereal cer[9], cir[9], cri[9];
```

```
    static real fxa, fya, fza, lat;
    extern /* Subroutine */ int mmk_();
    static doublereal cwr[9], sur, xyz[3], tmp2;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
/*
-----------------------------------------------------------------
-----c */
/* ---------------------------- missile state initialization module
----c */
/*
-----------------------------------------------------------------
-----c */
/*   initialize states and state derivatives */
    d_1 = dtr * (float)-90.;
    d_2 = latlp * dtr;
    d_3 = longlp * dtr;
    mmk_(&d_1, &c__1, &d_2, &c__2, &d_3, &c__3, cei);
    trans_(cei, cie);
/*
-----------------------------------------------------------------
------ */
/*  initialize processor inputs if not already initialized */
/* p2 */
    qa = (float)0.;
    mach = (float)0.;
/*  p5 */
    for (i = 1; i <= 3; ++i) {
      vrwm[i - 1] = (float)0.;
/* L10: */
    }
/*
-----------------------------------------------------------------
-----c */
/* ----------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L20:
/*
*********************************************************************
 */
/*                                                              * */
/*                            partition 1                         * */
/*                                                              * */
/*
*********************************************************************
 */
    send_real_32bit__(&mach);
    send_real_32bit__(&qa);
    send_real_32bit__(vrwm);
    send_real_32bit__(&vrwm[1]);
    send_real_32bit__(&vrwm[2]);
    d_1 = omegae * t;
    mmk_(&c_b6, &c__1, &c_b6, &c__2, &d_1, &c__3, cer);
    mmlxy_(cer, cie, cir);
    trans_(cir, cri);
/* ----------------- get parameters from main partition 1 thread
---------c */
    receive_real_64bit__(xyz);
    receive_real_64bit__(&xyz[1]);
```

```
    receive_real_64bit__(&xyz[2]);
    receive_real_64bit__(xyzd);
    receive_real_64bit__(&xyzd[1]);
    receive_real_64bit__(&xyzd[2]);
    receive_real_32bit__(cim);
    receive_real_32bit__(&cim[1]);
    receive_real_32bit__(&cim[2]);
    receive_real_32bit__(&cim[3]);
    receive_real_32bit__(&cim[4]);
    receive_real_32bit__(&cim[5]);
    receive_real_32bit__(&cim[6]);
    receive_real_32bit__(&cim[7]);
    receive_real_32bit__(&cim[8]);
    xyze[0] = cie[0] * xyz[0] + cie[3] * xyz[1] + cie[6] * xyz[2];
    xyze[1] = cie[1] * xyz[0] + cie[4] * xyz[1] + cie[7] * xyz[2];
    xyze[2] = cie[2] * xyz[0] + cie[5] * xyz[1] + cie[8] * xyz[2];
    vecrot_(xyze, cer, xyzr);
/*  calculate current latitude and longitude */
/* Computing 2nd power */
    d_2 = xyzr[0];
/* Computing 2nd power */
    d_3 = xyzr[1];
    d_1 = sqrt(d_2 * d_2 + d_3 * d_3);
    lat = arctan_(&xyzr[2], &d_1);
    long_ = arctan_(&xyzr[1], xyzr);
/* **********   start of winds subroutine
*******************************
c */
/*      call mmk(0.0d0,1,-lat,2,long,3,crw) */
/*      call trans(crw,cwr) */
    a = cos(-(doublereal)lat);
    b = sin(-(doublereal)lat);
    c = cos(long_);
    d = sin(long_);
    cwr[0] = a * c;
    cwr[1] = d;
    cwr[2] = b * c;
    cwr[3] = a * d;
    cwr[4] = c;
    cwr[5] = b * d;
    cwr[6] = b;
    cwr[7] = (float)0.;
    cwr[8] = a;
/* ----------- get masspr table look up values from other processors
----c */
    receive_real_32bit__(&vwind);
    receive_real_32bit__(&shear);
    receive_real_32bit__(&swdir);
    receive_real_32bit__(&cwdir);
/*      call vmk(shear,cwdir*vwind,swdir*vwind,vwwind) */
    vwwind[0] = shear;
    vwwind[1] = cwdir * vwind;
    vwwind[2] = swdir * vwind;
    vecrot_(vwwind, cwr, vrwind);
    vecrot_(vrwind, cri, viwind);
    vecsub_(xyzd, viwind, vrwi);
/*      call vecrot(vrwi,cim,vrwm) */
    vrwm[0] = cim[0] * vrwi[0] + cim[3] * vrwi[1] + cim[6] * vrwi[2];
    vrwm[1] = cim[1] * vrwi[0] + cim[4] * vrwi[1] + cim[7] * vrwi[2];
    vrwm[2] = cim[2] * vrwi[0] + cim[5] * vrwi[1] + cim[8] * vrwi[2];
/* Computing 2nd power */
    r_1 = vrwm[0];
/* Computing 2nd power */
    r_2 = vrwm[1];
```

```
/* Computing 2nd power */
    r_3 = vrwm[2];
    mvrwm = sqrt(r_1 * r_1 + r_2 * r_2 + r_3 * r_3);
/* **********  end of winds subroutine
*********************************
c */
/* ------------ calculate parameters from aero and start table lookups -
c
*/
    receive_real_32bit__(&vsnd);
    mach = mvrwm / vsnd;
/* Computing 2nd power */
    r_1 = vrwm[1];
/* Computing 2nd power */
    r_2 = vrwm[2];
    tmp1 = sqrt(r_1 * r_1 + r_2 * r_2);
    tmp2 = dabs(vrwm[0]);
    alfat = arctan_(&tmp1, &tmp2) / dtr;
    send_real_32bit__(&mach);
    send_real_32bit__(&alfat);
    receive_real_32bit__(&rhod2);
/* Computing 2nd power */
    d_1 = mvrwm;
    qa = d_1 * d_1 * rhod2;
/* -------------perform part of aero subroutine
-------------------------c */
    if (mvrwm <= 0. || t >= tstg2) {
      fxa = (float)0.;
      fya = (float)0.;
      fza = (float)0.;
      receive_real_32bit__(&ca);
      receive_real_32bit__(&cn);
    } else {
      if (abs(tmp1) > 1e-6) {
          cphia = vrwm[2] / tmp1;
          sphia = vrwm[1] / tmp1;
      } else {
          cphia = 1.;
          sphia = 0.;
      }
      if (t > tstg1) {
          sur = sref2;
      } else {
          sur = sref1;
      }
      qs = qa * sur;
      receive_real_32bit__(&ca);
      receive_real_32bit__(&cn);
      fxa = qs * ca;
      fya = -qs * cn * sphia;
      fza = -qs * cn * cphia;
    }
    send_real_32bit__(&fxa);
    send_real_32bit__(&fya);
    send_real_32bit__(&fza);
/*
*************************************************************************
 */
/*                                                                     * */
/*                          end of partition 1
 * */
/*                                                                     * */
/*
*************************************************************************
```

```
 */
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L20;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.8 Boost2a2.c

```
/* boost2a2.f -- translated by f2c (version of 3 February 1990
3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__1 = 1;
static integer c__2 = 2;
static integer c__3 = 3;
static integer c__12 = 12;
static integer c__13 = 13;
static integer c__14 = 14;
static integer c__15 = 15;
static integer c__16 = 16;
static integer c__17 = 17;
static integer c__18 = 18;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real phiicd = (float)0.;
    static real thticd = (float)-35.;
    static real psiicd = (float)0.;
    static real tfinal = (float)62.501;
    static real dtr = (float).017453292519943296;
    static real tmp1 = (float)0.;
    static real quatm = (float)1.;

    static real told, quat[4];
    static integer itst;
    extern doublereal sparctan_();
    extern /* Subroutine */ int spintegi_  ;
    static integer i;
    static real p, q, r, t, quatd[4];
    extern /* Subroutine */ int fvdot_(), spmmk_();
    static real tstep;
    extern /* Subroutine */ int receive_real_32bit__(), bxi2fv_(),
fv2bxi_();
    static real pd, qd, rd;
    extern doublereal arcsin_();
    extern /* Subroutine */ int send_real_32bit__();
    static real cim[9], cmi[9], phi, psi;
    extern /* Subroutine */ int vmk_();
    static real tht, pqr[3];
    extern /* Subroutine */ int spinteg_(), sptrans_();
    static real tmp2;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    for (i = 1; i <= 3; ++i) {
      pqr[i - 1] = (float)0.;
/* L10: */
    }
    phi = phiicd * dtr;
```

```
    tht = thticd * dtr;
    psi = psiicd * dtr;
    spmmk_(&phi, &c__1, &tht, &c__2, &psi, &c__3, cim);
    sptrans_(cim, cmi);
/*
-----------------------------------------------------------------
-----c */
/* ---------------------------- missile state initialization module
----c */
/*
-----------------------------------------------------------------
-----c */
/*   initialize states and state derivatives */
    bxi2fv_(&quatm, cmi, quat);
    pd = (float)0.;
    qd = (float)0.;
    rd = (float)0.;
    fvdot_(pqr, &tmp1, quat, quatd);
    spintegi_(pqr, &pd, &t, &c__12);
    spintegi_(&pqr[1], &qd, &t, &c__13);
    spintegi_(&pqr[2], &rd, &t, &c__14);
    spintegi_(quat, quatd, &t, &c__15);
    spintegi_(&quat[1], &quatd[1], &t, &c__16);
    spintegi_(&quat[2], &quatd[2], &t, &c__17);
    spintegi_(&quat[3], &quatd[3], &t, &c__18);
    p = pqr[0];
    q = pqr[1];
    r = pqr[2];
    itst = 0;
/*
-----------------------------------------------------------------
-----c */
/* ---- ------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L20:
/*
*****  ***************************************************************
 */
/*                                                          * */
/*                              partition 1                            * */
/*                                                          * */
/*
***************************************************************************
 */
    send_real_32bit__(cim);
    send_real_32bit__(&cim[1]);
    send_real_32bit__(&cim[2]);
    send_real_32bit__(&cim[3]);
    send_real_32bit__(&cim[4]);
    send_real_32bit__(&cim[5]);
    send_real_32bit__(&cim[6]);
    send_real_32bit__(&cim[7]);
    send_real_32bit__(&cim[8]);
/*      Added for graphics program */
    send_real_32bit__(&phi);
    send_real_32bit__(&tht);
    send_real_32bit__(&psi);
    receive_real_32bit__(&pd);
    receive_real_32bit__(&qd);
    receive_real_32bit__(&rd);
    if (itst == 0) {
```

```
      itst = 1;
    } else {
      spinteg_(&p, &pd, &told, &c__12);
      spinteg_(&q, &qd, &told, &c__13);
      spinteg_(&r, &rd, &told, &c__14);
    }
    p += delt * pd;
    q += delt * qd;
    r += delt * rd;
    quat[0] += delt * quatd[0];
    quat[1] += delt * quatd[1];
    quat[2] += delt * quatd[2];
    quat[3] += delt * quatd[3];
/* ---------section of missil subroutine that finds phi, tht, and
psi----c */
    vmk_(&p, &q, &r, pqr);
    tmp2 = (float)0.;
    fvdot_(pqr, &tmp2, quat, quatd);
    fv2bxi_(quat, &tmp2, cmi);
    sptrans_(cmi, cim);
    phi = sparctan_(&cim[7], &cim[8]);
    tht = -(doublereal)arcsin_(&cim[6]);
    psi = sparctan_(&cim[3], cim);
/*
-------------------------------------------------------------------
-----c */
/*                              missile state integration module
c
*/
/*
-------------------------------------------------------------------
-----c */
    spinteg_(quat, quatd, &t, &c__15);
    spinteg_(&quat[1], &quatd[1], &t, &c__16);
    spinteg_(&quat[2], &quatd[2], &t, &c__17);
    spinteg_(&quat[3], &quatd[3], &t, &c__18);
/*
**********************************************************************
 */
/*                                                          * */
/*                        end of partition 1
 * */
/*                                                          * */
/*
**********************************************************************
 */
    told = t;
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L20;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.9 Boost2a3.c

```
/* boost2a3.f -- translated by f2c (version of 3 February 1990
3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__12 = 12;
static integer c__13 = 13;
static integer c__14 = 14;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static doublereal delt = .001;
    static doublereal xlnch = 3.;
    static doublereal gmu = 1.4052477e16;
    static doublereal tfinal = 62.501;
    static doublereal rade = 20898908.;
    static integer nclear = 0;
    static integer imis = 0;

    /* System generated locals */
    doublereal d_1;

    /* Local variables */
    extern /* Subroutine */ int magt_();
    static real frcx, mass, frcy, frcz, mrcx, mrcy, mrcz;
    extern /* Subroutine */ int spintegi_();
    static doublereal mxyz, uxyz[3];
    static integer i;
    static real p, q, r;
    static doublereal t, tstep;
    extern /* Subroutine */ int mvbys_(), receive_real_32bit__(),
          receive_real_64bit__();
    static doublereal gb[3];
    static real cg[3], pd, qd, rd;
    static doublereal gr[3], fx, fy, fz, mx, my, mz;
    extern /* Subroutine */ int send_real_32bit__();
    static doublereal xyzlch[3];
    static real cim[9], cmi[9], fxa, fya, fza;
    static doublereal mxa, mya, mza, mgr;
    static real xcp, fxt, fyt, fzt, pqr[3], spt, ixx, mxt, iyy, myt,
izz, mzt;

    extern /* Subroutine */ int spinteg_();
    static doublereal xyz[3];
    extern /* Subroutine */ int sptrans_();

/* $include(':pfp:include/target.for') */
/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    for (i = 1; i <= 3; ++i) {
      pqr[i - 1] = (float)0.;
/* L10: */
    }
```

```
/*
-----------------------------------------------------------------
-----c */
/* ----------------------------- missile state initialization module
----c */
/*
-----------------------------------------------------------------
-----c */
/*   initialize states and state derivatives */
    pd = (float)0.;
    qd = (float)0.;
    rd = (float)0.;
    spt = t;
    spintegi_(pqr, &pd, &spt, &c__12);
    spintegi_(&pqr[1], &qd, &spt, &c__13);
    spintegi_(&pqr[2], &rd, &spt, &c__14);
    p = pqr[0];
    q = pqr[1];
    r = pqr[2];
    for (i = 1; i <= 3; ++i) {
      gr[i - 1] = (float)0.;
/* L20: */
    }
/*
-----------------------------------------------------------------
-----c */
/* ----------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L30:
/*
*************************************************************************
 */
/*                                                          * */
/*                            partition 1                               * */
/*                                                          * */
/*
*************************************************************************
 */
/* ------------- send parameters to partitions 3, 4, and 5
-------------c */
    send_real_32bit__(&p);
    send_real_32bit__(&q);
    send_real_32bit__(&r);
    send_real_32bit__(&pd);
    send_real_32bit__(&qd);
    send_real_32bit__(&rd);
    receive_real_32bit__(&mass);
    receive_real_64bit__(xyz);
    receive_real_64bit__(&xyz[1]);
    receive_real_64bit__(&xyz[2]);
    receive_real_32bit__(cim);
    receive_real_32bit__(&cim[1]);
    receive_real_32bit__(&cim[2]);
    receive_real_32bit__(&cim[3]);
    receive_real_32bit__(&cim[4]);
    receive_real_32bit__(&cim[5]);
    receive_real_32bit__(&cim[6]);
    receive_real_32bit__(&cim[7]);
    receive_real_32bit__(&cim[8]);
/* ---------- receive parameters from partition #2
----------------------c */
```

```
    receive_real_32bit__(&fxt);
    receive_real_32bit__(&fyt);
    receive_real_32bit__(&fzt);
    receive_real_32bit__(&mxt);
    receive_real_32bit__(&myt);
    receive_real_32bit__(&mzt);
    receive_real_32bit__(&frcx);
    receive_real_32bit__(&frcy);
    receive_real_32bit__(&frcz);
    receive_real_32bit__(&mrcx);
    receive_real_32bit__(&mrcy);
    receive_real_32bit__(&mrcz);
/* ---------------------------- mass properties module
------------------c */
    receive_real_32bit__(cg);
    receive_real_32bit__(&cg[1]);
    receive_real_32bit__(&cg[2]);
    receive_real_32bit__(&ixx);
    receive_real_32bit__(&iyy);
    receive_real_32bit__(&izz);
    p += delt * pd;
    q += delt * qd;
    r += delt * rd;
/* ---------------------------- vehicle states module
------------------c */
/*
-----------------------------------------------------------------------
------ */
    if (imis == 0) {
      sptrans_(cim, cmi);
      xyzlch[0] = xlnch * cmi[0] + rade;
      xyzlch[1] = xlnch * cmi[1];
      xyzlch[2] = xlnch * cmi[2];
      imis = 1;
    }
    magt_(xyz, &mxyz, uxyz);
/* Computing 2nd power */
    d_1 = mxyz;
    mgr = gmu / (d_1 * d_1);
    d_1 = -mgr;
    mvbys_(&d_1, uxyz, gr);
/*          CALL vecrot(gr, cim, gb) */
    gb[0] = cim[0] * gr[0] + cim[3] * gr[1] + cim[6] * gr[2];
    gb[1] = cim[1] * gr[0] + cim[4] * gr[1] + cim[7] * gr[2];
    gb[2] = cim[2] * gr[0] + cim[5] * gr[1] + cim[8] * gr[2];
/* ----------------- section of aero subroutine to --------------------c
*/
/* ----------------- calculate mxa,mya, and mza from fxa,fya,and fza --c
*/
    receive_real_32bit__(&fxa);
    receive_real_32bit__(&fya);
    receive_real_32bit__(&fza);
    receive_real_32bit__(&xcp);
    mxa = fya * cg[2] - fza * cg[1];
    mya = -(doublereal)fxa * cg[2] + fza * (cg[0] - xcp);
    mza = fxa * cg[1] - fya * (cg[0] - xcp);
/* ---------------- section of missil subroutine to find ---------------
c
*/
/* ----------------  pd, qd, and rd                        ---------------
c
*/
    fx = fxt + fxa + frcx;
    fy = fyt + fya + frcy;
```

```
    fz = fzt + fza + frcz;
    mx = mxa + mxt + mrcx;
    my = mya + myt + mrcy;
    mz = mza + mzt + mrcz;
    if (nclear == 1) {
      pd = (float)0.;
      qd = my / iyy + r * p * ((izz - ixx) / iyy);
      rd = mz / izz + p * q * ((ixx - iyy) / izz);
    } else if (fx / mass <= abs(gb[0])) {
      pd = (float)0.;
      qd = (float)0.;
      rd = (float)0.;
    } else if (xyz[0] <= xyzlch[0] && xyz[1] <= xyzlch[1] && xyz[2] <=
xyzlch[
          2]) {
      pd = (float)0.;
      qd = (float)0.;
      rd = (float)0.;
    } else {
      nclear = 1;
/*        call output_message( %val(character_08bit), */
/*     .    ' missile has cleared the launcher' ) */
/*        call output_nl */
      pd = (float)0.;
      qd = my / iyy + r * p * ((izz - ixx) / iyy);
      rd = mz / izz + p * q * ((ixx - iyy) / izz);
    }
/*
-----------------------------------------------------------------
-----c */
/*                              missile state integration module
c
*/
/*
-----------------------------------------------------------------
-----c */
    spt = t;
    spinteg_(&p, &pd, &spt, &c__12);
    spinteg_(&q, &qd, &spt, &c__13);
    spinteg_(&r, &rd, &spt, &c__14);
/*
**********************************************************************
 */
/*                                                              * */
/*                         end of partition 1
 * */
/*                                                              * */
/*
**********************************************************************
 */
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L30;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.10 Boost2b.c

```
/* boost2b.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real tstg2 = (float)62.5;
    static real tfrac = (float)23001.;
    static real dtfru = (float)5.;
    static real tfrcs = (float)23.001;

    static real mach, dlpc;
    static integer lenf[4];
    static real dlyc, frcx, frcy, frcz, mrcx, mrcy, mrcz;
    static integer i, j;
    static real t;
    extern /* Subroutine */ int fracs_();
    static real mdotf, tstep;
    extern /* Subroutine */ int receive_real_32bit__();
    static real cg[3], qa, pm[3], malpha, sq, sr, mdltfr;
    extern /* Subroutine */ int frcthr_(), send_real_32bit__();
    static real thf[40] /* was [10][4] */, tmf[40]    /* was [10][4] */;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    for (i = 1; i <= 10; ++i) {
      for (j = 1; j <= 4; ++j) {
          tmf[i + j * 10 - 11] = (float)0.;
          thf[i + j * 10 - 11] = (float)0.;
/* L10: */
      }
/* L20: */
    }
    for (i = 1; i <= 4; ++i) {
      lenf[i - 1] = 0;
/* L30: */
    }
/*
-----------------------------------------------------------------
-----c */
/* ---------------------------- missile state initialization module
----c */
/* p1 */
    frcx = (float)0.;
    frcy = (float)0.;
    frcz = (float)0.;
    mrcx = (float)0.;
    mrcy = (float)0.;
    mrcz = (float)0.;
    mdotf = (float)0.;
/*
-----------------------------------------------------------------
-----c */
```

```
/* ---------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L40:
/*
*************************************************************************
 */
/*                                                          * */
/*                            partition 2                             * */
/*                                                          * */
/*
*************************************************************************
 */
/* --------------- send parameters to partition #1
----------------------c */
    send_real_32bit__(&frcx);
    send_real_32bit__(&frcy);
    send_real_32bit__(&frcz);
    send_real_32bit__(&mrcx);
    send_real_32bit__(&mrcy);
    send_real_32bit__(&mrcz);
    send_real_32bit__(&mdotf);
/* ---------- receive parameters from partition #1
----------------------c */
    receive_real_32bit__(cg);
    receive_real_32bit__(&cg[1]);
    receive_real_32bit__(&cg[2]);
/* ---------- receive parameters from partition #1
----------------------c */
    receive_real_32bit__(&mach);
    receive_real_32bit__(&qa);
/* ---------- receive parameters from partition #3,4, and 5
-------------c */
    receive_real_32bit__(&dlpc);
    receive_real_32bit__(&dlyc);
    receive_real_32bit__(&sq);
    receive_real_32bit__(&sr);
    receive_real_32bit__(&mdltfr);
    receive_real_32bit__(&malpha);
    receive_real_32bit__(pm);
    receive_real_32bit__(&pm[1]);
    receive_real_32bit__(&pm[2]);
/* ---------------------------- fracs thruster response module
---------c */
    if (t >= tfrcs && t < tstg2) {
      frcthr_(&t, cg, &mach, &qa, tmf, thf, lenf, &frcx, &frcy, &frcz, &
            mrcx, &mrcy, &mrcz, &mdotf);
/* --------------------------- fracs logic module
----------------------c */
      if (tstep >= tfrac) {
          fracs_(&t, &dlpc, &dlyc, &sq, &sr, &mdltfr, &malpha, pm, tmf,
thf,
                 lenf);
          tfrac += dtfru;
      }
    } else {
      frcx = (float)0.;
      frcy = (float)0.;
      frcz = (float)0.;
      mrcx = (float)0.;
      mrcy = (float)0.;
      mrcz = (float)0.;
```

```
      mdotf = (float)0.;
    }
/*
*****************************************************************************
 */
/*                                                                    * */
/*                          end of partition 2
 * */
/*                                                                    * */
/*
*****************************************************************************
 */
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L40;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.11 Boost2b1.c

```
/* boost2b1.f -- translated by f2c (version of 3 February 1990
3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__19 = 19;
static integer c__20 = 20;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real dlyic = (float)0.;
    static real tfinal = (float)62.501;
    static real tstg1 = (float)23.;
    static real tinhib = (float).35;
    static real pmax = (float).13963;
    static real dlpic = (float)0.;

    /* System generated locals */
    real r_1, r_2;

    /* Local variables */
    static real cmmd[2], dlpd, dlyd;
    extern /* Subroutine */ int spintegi_();
    static real t, mdott, press, tstep;
    extern /* Subroutine */ int receive_real_32bit__();
    static real cg[3], totdel;
    extern /* Subroutine */ int send_real_32bit__(), bthrst_();
    static real dlp;
    extern /* Subroutine */ int ncu_();
    static real dly, fxt, fyt, fzt, mxt, myt, mzt;
    extern /* Subroutine */ int spinteg_();

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
/*
-------------------------------------------------------------------
-----c */
/* ---------------------------- missile state initialization module
----c */
/*
-------------------------------------------------------------------
-----c */
/*   initialize states and state derivatives */
    dlp = dlpic;
    dly = dlyic;
    dlpd = (float)0.;
    dlyd = (float)0.;
    spintegi_(&dlp, &dlpd, &t, &c__19);
    spintegi_(&dly, &dlyd, &t, &c__20);
/*
-------------------------------------------------------------------
------ */
```

```
/*  initialize processor inputs if not already initialized */
/* p1 */
    fxt = (float)0.;
    fyt = (float)0.;
    fzt = (float)0.;
    mxt = (float)0.;
    myt = (float)0.;
    mzt = (float)0.;
    mdott = (float)0.;
/*
-----------------------------------------------------------------
-----c */
/* ----------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L10:
/*
*************************************************************************
 */
/*                                                              * */
/*                              partition 2                             * */
/*                                                              * */
/*
*************************************************************************
 */
/* --------------- send parameters to partition #1
----------------------c */
    send_real_32bit__(&fxt);
    send_real_32bit__(&fyt);
    send_real_32bit__(&fzt);
    send_real_32bit__(&mxt);
    send_real_32bit__(&myt);
    send_real_32bit__(&mzt);
    send_real_32bit__(&mdott);
/* ---------- receive parameters from partition #1
----------------------c */
    receive_real_32bit__(cg);
    receive_real_32bit__(&cg[1]);
    receive_real_32bit__(&cg[2]);
/* ---------- receive parameters from partition #1
----------------------c */
    receive_real_32bit__(&press);
/* ---------- receive parameters from partition #3,4, and 5
-------------c */
    receive_real_32bit__(cmmd);
    receive_real_32bit__(&cmmd[1]);
    if (t <= tstg1) {
      dlp += delt * dlpd;
      dly += delt * dlyd;
/* Computing 2nd power */
      r_1 = dlp;
/* Computing 2nd power */
      r_2 = dly;
      totdel = sqrt(r_1 * r_1 + r_2 * r_2);
      if (totdel > pmax) {
          dlp = dlp * pmax / totdel;
          dly = dly * pmax / totdel;
      }
    }
/* ---------------------------- boosters module
-----------------------c */
```

```
    bthrst_(&t, cg, &press, &dlp, &dly, &fxt, &fyt, &fzt, &mxt, &myt,
&mzt, &
            mdott);
/* ------------------------------ nozzle control unit module
-------------c */
    if (t <= tstg1) {
      if (t > tinhib) {
          ncu_(&dlp, &dly, cmmd, &dlpd, &dlyd);
      } else {
          dlpd = (float)0.;
          dlyd = (float)0.;
      }
      spinteg_(&dlp, &dlpd, &t, &c__19);
      spinteg_(&dly, &dlyd, &t, &c__20);
/* Computing 2nd power */
      r_1 = dlp;
/* Computing 2nd power */
      r_2 = dly;
      totdel = sqrt(r_1 * r_1 + r_2 * r_2);
      if (totdel > pmax) {
          dlp = dlp * pmax / totdel;
          dly = dly * pmax / totdel;
      }
    } else {
      dlp = (float)0.;
      dly = (float)0.;
    }
/*
*************************************************************************
 */
/*                                                                    * */
/*                          end of partition 2
 * */
/*                                                                    * */
/*
*************************************************************************
 */
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.12 Boost2c.c

```
/* boost2c.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real dtr = (float).017453292519943296;
    static real thticd = (float)-35.;
    static real psiicd = (float)0.;
    static real vp1 = (float)13770.;
    static real us0d = (float)-22.;
    static real tfinal = (float)62.501;
    static real tstg2 = (float)62.5;
    static real tst2on = (float)22.995;
    static real dtbgu = (float)5.;

    /* System generated locals */
    real r_1, r_2, r_3;

    /* Local variables */
    static real delu, delv, delw, spsi, tgpu, stht, dtmp1;
    static integer i;
    static real t;
    extern /* Subroutine */ int bguid_();
    static real delxd, delyd, delzd;
    extern /* Subroutine */ int navig_();
    static real attlm, psier, thter, tlgpu, tstep;
    extern /* Subroutine */ int receive_real_32bit__();
    static real ac[3], at[3], pg[3], dt, gr[3], pm[3], sq, sr, delphi,
us[3],
          vw[3], delpsi, deltht;
    extern /* Subroutine */ int bsteer_(), send_real_32bit__();
    static real mvrdot, pgd[3], usd[3], vwd[3], pqr[3], mvr, mvs,
uvs[3],
          ti2m[9];

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    for (i = 1; i <= 3; ++i) {
      pqr[i - 1] = (float)0.;
      usd[i - 1] = (float)0.;
      pgd[i - 1] = (float)0.;
      vwd[i - 1] = (float)0.;
      vw[i - 1] = (float)0.;
/* L10: */
    }
    mvrdot = (float)0.;
    stht = thticd * dtr;
    spsi = psiicd * dtr;
    sq = pqr[1];
    sr = pqr[2];
    us[0] = cos(spsi) * cos(us0d * dtr);
    us[1] = sin(spsi) * cos(us0d * dtr);
    us[2] = -(doublereal)sin(us0d * dtr);
```

```
    pg[0] = cos(spsi) * cos(stht);
    pg[1] = sin(spsi) * cos(stht);
    pg[2] = -(doublereal)sin(stht);
/*
------ ----------------------------------------------------------- ----
-----c */
/* ------ ------------------------ missile state initialization module
----c */
/*
------------ ------------------------------------------------------------
------ */
/*  initialize processor inputs if not already initialized */
    for (i = 1; i <= 3; ++i) {
      pm[i - 1] = (float)0.;
/* L20: */
    }
    delu = (float)0.;
    delv = (float)0.;
    delw = (float)0.;
    for (i = 1; i <= 3; ++i) {
      at[i - 1] = (float)0.;
/* L30: */
    }
    for (i = 1; i <= 9; ++i) {
      ti2m[i - 1] = (float)0.;
/* L40: */
    }
    mvr = vp1;
/*  initialization routine */
    tgpu = (float)0.;
    tlgpu = (float)0.;
/*
------------ ------------------------------------ ----------------
------c */
/* ------------------------------ main execution loop
--------------------c */
/*
------------------------------------------------------------------------
-----c */
L50:
/* --------------- send parameters to partition #2
-----------------------c */
    send_real_32bit__(at);
    send_real_32bit__(&at[1]);
    send_real_32bit__(&at[2]);
    send_real_32bit__(&delxd);
    send_real_32bit__(&delyd);
    send_real_32bit__(&delzd);
    send_real_32bit__(pm);
    send_real_32bit__(&pm[1]);
    send_real_32bit__(&pm[2]);
/* --------------- receive parameters from partition #1
-----------------c */
    receive_real_32bit__(gr);
    receive_real_32bit__(&gr[1]);
    receive_real_32bit__(&gr[2]);
    receive_real_32bit__(&mvs);
    receive_real_32bit__(uvs);
    receive_real_32bit__(&uvs[1]);
    receive_real_32bit__(&uvs[2]);
    receive_real_32bit__(&delphi);
    receive_real_32bit__(&deltht);
    receive_real_32bit__(&delpsi);
    receive_real_32bit__(&delu);
```

```
    receive_real_32bit__(&delv);
    receive_real_32bit__(&delw);
/*
*************************************************************************
 */
/*                                                                       * */
/*                              partition 4                                * */
/*                                                                       * */
/*
*************************************************************************
 */
/* ------------------------------ navigation module
----------------------c */
    navig_(&delphi, &deltht, &delpsi, &delu, &delv, &delw, gr, &t, &sq,
&sr,
          ti2m, at, &delxd, &delyd, &delzd);
/*  integrate performance velocity remaining using navigation output */
    if (t < tst2on || t >= tstg2) {
      mvrdot = (float)0.;
    } else {
/* Computing 2nd power */
      r_1 = at[0];
/* Computing 2nd power */
      r_2 = at[1];
/* Computing 2nd power */
      r_3 = at[2];
      mvrdot = -(doublereal)sqrt(r_1 * r_1 + r_2 * r_2 + r_3 * r_3);
    }
    mvr += delt * mvrdot;
    if (mvr < (float)0.) {
      mvr = (float)0.;
    }
/*
*************************************************************************
 */
/*                                                                       * */
/*                          end of partition 4
 * */
/*                                                                       * */
/*
*************************************************************************
 */
/*
*************************************************************************
 */
/*                                                                       * */
/*                              partition 5                                * */
/*                                                                       * */
/*
*************************************************************************
 */
/*--------------------on board guidance
processing--------------------------c*/
    if (tstep >= tgpu) {
      tgpu += dtbgu;
      dt = t - tlgpu;
      tlgpu = t;
/*  integrate guidance states from last pass through */
      us[0] += dt * usd[0];
      us[1] += dt * usd[1];
      us[2] += dt * usd[2];
      pg[0] += dt * pgd[0];
      pg[1] += dt * pgd[1];
      pg[2] += dt * pgd[2];
```

```
      vw[0] += dt * vwd[0];
      vw[1] += dt * vwd[1];
      vw[2] += dt * vwd[2];
/* Computing 2nd power */
      r_1 = pg[0];
/* Computing 2nd power */
      r_2 = pg[1];
/* Computing 2nd power */
      r_3 = pg[2];
      dtmp1 = sqrt(r_1 * r_1 + r_2 * r_2 + r_3 * r_3);
      pg[0] /= dtmp1;
      pg[1] /= dtmp1;
      pg[2] /= dtmp1;
/* ------------------------------ boost steering module
------------------c */
      if (t < tstg2) {
          bsteer_(&t, us, uvs, &mvs, &mvr, at, usd, ac);
/* ------------------------------ boost guidance module
------------------c */
          bguid_(&t, at, ac, ti2m, pg, vw, pgd, vwd, &psier, &thter,
pm);
      } else {
          receive_real_32bit__(&attlm);
      }
      if (t >= tstg2) {
          usd[0] = (float)0.;
          usd[1] = (float)0.;
          usd[2] = (float)0.;
          pgd[0] = (float)0.;
          pgd[1] = (float)0.;
          pgd[2] = (float)0.;
          vwd[0] = (float)0.;
          vwd[1] = (float)0.;
          vwd[2] = (float)0.;
      }
    } else {
      receive_real_32bit__(&attlm);
    }
    send_real_32bit__(&psier);
    send_real_32bit__(&thter);
    send_real_32bit__(&sq);
    send_real_32bit__(&sr);
/*
*****************************************************************************
 */
/*                                                                  * */
/*                            end of partition 5
 * */
/*                                                                  * */
/*
*****************************************************************************
 */
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L50;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.13 Boost2c1.c

```
/* boost2c1.f -- translated by f2c (version of 3 February 1990
3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real delu = (float)0.;
    static real delv = (float)0.;
    static real delw = (float)0.;

    extern /* Subroutine */ int gyro_(), accel_();
    static real p, q, r, t, tstep;
    extern /* Subroutine */ int receive_real_32bit__();
    static real cg[3], pd, qd, rd, ud, vd, wd, delphi, delpsi, deltht,
pulsea[
          3];
    extern /* Subroutine */ int send_real_32bit__();
    static real pulseg[3];
    extern /* Subroutine */ int imupro_();

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
/*
-----------------------------------------------------------------
-----c */
/* ----------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L10:
/* -------------- receive parameters from partition #1
------------------c */
    receive_real_32bit__(cg);
    receive_real_32bit__(&cg[1]);
    receive_real_32bit__(&cg[2]);
    receive_real_32bit__(&p);
    receive_real_32bit__(&q);
    receive_real_32bit__(&r);
    receive_real_32bit__(&ud);
    receive_real_32bit__(&vd);
    receive_real_32bit__(&wd);
    receive_real_32bit__(&pd);
    receive_real_32bit__(&qd);
    receive_real_32bit__(&rd);
/*
*****************************************************************
 */
/*                                                          * */
/*                          partition 3                             * */
/*                                                          * */
```

```
/*
*************************************************************************
 */
/* ----------------------------- inertial measurement update
------------c */
    gyro_(&p, &q, &r, &t, pulseg);
/* ----------------------------- inertial measurement update
------------c */
    accel_(&ud, &vd, &wd, &p, &q, &r, &pd, &qd, &rd, cg, &t, pulsea);
/* ----------------------------- imu processor module
-------------------c */
    imupro_(pulsea, pulseg, &delphi, &deltht, &delpsi, &delu, &delv,
&delw);
    send_real_32bit__(&delphi);
    send_real_32bit__(&deltht);
    send_real_32bit__(&delpsi);
    send_real_32bit__(&delu);
    send_real_32bit__(&delv);
    send_real_32bit__(&delw);
/*
*************************************************************************
 */
/*                                                                    * */
/*                            end of partition 3
 * */
/*                                                                    * */
/*
*************************************************************************
 */
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.14 Boost2c2.c

```
/* boost2c2.f -- translated by f2c (version of 3 February 1990
3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__1 = 1;
static integer c__2 = 2;
static integer c__3 = 3;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static doublereal dtr = .017453292519943296;
    static doublereal latlp = 0.;
    static doublereal longlp = 0.;
    static real vp1 = (float)13770.;
    static doublereal tfinal = 62.501;
    static real tstg2 = (float)62.5;
    static real tst2on = (float)22.995;
    static real dtcvu = (float)50.;

    /* System generated locals */
    real r_1, r_2, r_3;
    doublereal d_1, d_2, d_3;

    /* Local variables */
    static doublereal rmir[3], vmir[3], xyze[3];
    static integer i;
    static real t, delxd, delyd, delzd, tcorv;
    static doublereal tstep, xyzed[3];
    extern /* Subroutine */ int receive_real_32bit__();
    static real at[3], vg[3];
    extern /* Subroutine */ int corvel_(), send_real_32bit__(),
incorv_(),
            vecrot_();
    static integer istart;
    static real mvrdot;
    static doublereal cei[9];
    extern /* Subroutine */ int mmk_();
    static real mvr, mvs, uvs[3], vtt[3];

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    for (i = 1; i <= 3; ++i) {
      xyze[i - 1] = (float)0.;
      xyzed[i - 1] = (float)0.;
      vtt[i - 1] = (float)0.;
      vg[i - 1] = (float)0.;
/* L10: */
    }
    xyze[0] = (float)20898908.;
    vg[0] = (float)5e3;
    vg[2] = (float)9350.;
```

```
    mvrdot = (float)0.;
/*
-----------------------------------------------------------------
-----c */
/* ---------------------------- missile state initialization module
----c */
/*
-----------------------------------------------------------------
-----c */
/*   initialize states and state derivatives */
    d_1 = dtr * (float)-90.;
    d_2 = latlp * dtr;
    d_3 = longlp * dtr;
    mmk_(&d_1, &c__1, &d_2, &c__2, &d_3, &c__3, cei);
    vecrot_(xyzed, cei, vmir);
    vecrot_(xyze, cei, rmir);
    incorv_(vg, rmir, vmir, &mvs, uvs);
    mvr = vpl;
    tcorv = (float)0.;
    istart = 0;
/*
-----------------------------------------------------------------
-----c */
/* ---------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L20:
/*
*****************************************************************
 */
/*                                                          * */
/*                              partition 4                       * */
/*                                                          * */
/*
*****************************************************************
 */
    send_real_32bit__(&mvs);
    send_real_32bit__(uvs);
    send_real_32bit__(&uvs[1]);
    send_real_32bit__(&uvs[2]);
    receive_real_32bit__(at);
    receive_real_32bit__(&at[1]);
    receive_real_32bit__(&at[2]);
    receive_real_32bit__(&delxd);
    receive_real_32bit__(&delyd);
    receive_real_32bit__(&delzd);
    if (istart == 0) {
      istart = 1;
    } else {
      rmir[0] += (vmir[0] + delxd * .5) * delt;
      rmir[1] += (vmir[1] + delyd * .5) * delt;
      rmir[2] += (vmir[2] + delzd * .5) * delt;
      vmir[0] += delxd;
      vmir[1] += delyd;
      vmir[2] += delzd;
/*  integrate performance velocity remaining using navigation output
*/
      if (t - delt < tst2on || t - delt >= tstg2) {
          mvrdot = (float)0.;
      } else {
/* Computing 2nd power */
          r_1 = at[0];
```

```
/* Computing 2nd power */
          r_2 = at[1];
/* Computing 2nd power */
          r_3 = at[2];
          mvrdot = -(doublereal)sqrt(r_1 * r_1 + r_2 * r_2 + r_3 * r_3);
        }
        mvr += delt * mvrdot;
        if (mvr < (float)0.) {
          mvr = (float)0.;
        }
/*  integrate gravity compensated acceleration */
        vtt[0] += delt * at[0];
        vtt[1] += delt * at[1];
        vtt[2] += delt * at[2];
      }
/* ------------------------------ correlated velocity module
-------------c */
      if (tstep >= tcorv) {
        corvel_(&mvr, &t, vtt, rmir, vmir, vg, &mvs, uvs);
        tcorv += dtcvu;
      }
/*
*********************************************************************
 */
/*                                                              * */
/*                          end of partition 4
 * */
/*                                                              * */
/*
*********************************************************************
 */
/* increment time */
      tstep += (float)1.;
      t = tstep * delt;
      if (t < tfinal) {
        goto L20;
      }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.15 Cg123.c

```
/* cg123.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__20 = 20;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real cgz[20] = {
(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,
          (float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)
          0.,(float)0.,(float)0. };
    static real cg[3] = { (float)0.,(float)0.,(float)0. };
    static integer itable = 0;
    static real tfinal = (float)62.501;
    static real masst1[20] = {
(float)2.403,(float)2.468,(float)2.524,(float)

2.724,(float)3.802,(float)4.231,(float)6.33,(float)8.631,(float)

11.024,(float)13.489,(float)14.932,(float)17.266,(float)18.955,(
          float)19.457,(float)22.121,(float)25.628,(float)30.806,(float)
          36.5,(float)40.889,(float)43.939 };
    static real cgx[20] = { (float)-2.628,(float)-2.628,(float)-
2.628,(float)
          -2.628,(float)-3.882,(float)-4.097,(float)-4.7,(float)-4.987,(
          float)-5.176,(float)-5.319,(float)-5.405,(float)-5.451,(float)
          -5.512,(float)-5.53,(float)-6.246,(float)-6.857,(float)-
7.473,(
          float)-7.922,(float)-8.222,(float)-8.425 };
    static real cgy[20] = {
(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,
          (float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)
          0.,(float)0.,(float)0. };

    static real mass, t;
    extern /* Subroutine */ int table_();
    static real tstep;
    extern /* Subroutine */ int receive_real_32bit__(),
send_real_32bit__();

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    send_real_32bit__(cg);
    send_real_32bit__(&cg[1]);
    send_real_32bit__(&cg[2]);
    receive_real_32bit__(&mass);
```

```
    table_(masst1, cgx, &mass, cg, &c__20, &itable);
    table_(masst1, cgy, &mass, &cg[1], &c__20, &itable);
    table_(masst1, cgz, &mass, &cg[2], &c__20, &itable);
    send_real_32bit__(cg);
    send_real_32bit__(&cg[1]);
    send_real_32bit__(&cg[2]);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.16 Cne.c

```
/* cne.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static real c_b2 = (float)4.;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static integer icnale = 0;
    static integer icnm2e = 0;
    static integer icna2e = 0;
    static real tapu = (float)0.;
    static real dtapu = (float)5.;
    static real tfinal = (float)62.501;
    static real tstg1 = (float)23.;
    static real tstg2 = (float)62.5;
    static real cnale[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)

160.,(float)170.,(float)180.,(float)0.,(float)0.,(float).1943,(

float).4861,(float).6991,(float)1.031,(float)1.8471,(float)2.8683,
          (float)5.198,(float)6.7229,(float)6.9385,(float)6.7289,(float)

5.1903,(float)2.8804,(float).8334,(float).2054,(float)0.,(float)

.5,(float)0.,(float).2016,(float).4596,(float).7148,(float)1.0404,
          (float)1.8189,(float)2.877,(float)5.1897,(float)6.7274,(float)

6.9327,(float)6.7164,(float)5.2139,(float)2.8484,(float).7949,(
          float).2014,(float)0.,(float).8,(float)0.,(float).2108,(float)
          .4888,(float).7689,(float)1.1237,(float)1.9773,(float)3.0839,(
          float)5.6104,(float)7.2627,(float)7.4829,(float)7.2585,(float)

5.6183,(float)3.0957,(float).8721,(float).2244,(float)0.,(float)

.9,(float)0.,(float).2453,(float).5632,(float).8506,(float)1.2693,

(float)2.1842,(float)3.3373,(float)6.0308,(float)7.7796,(float)

8.0298,(float)7.7874,(float)6.0173,(float)3.3226,(float).9462,(
          float).2334,(float)0.,(float)1.1,(float)0.,(float).259,(float)
          .6233,(float).9733,(float)1.4721,(float)2.5275,(float)3.9615,(
          float)7.1644,(float)9.2488,(float)9.5353,(float)9.2653,(float)

7.1618,(float)3.9405,(float)1.1188,(float).2778,(float)0.,(float)

1.2,(float)0.,(float).249,(float).629,(float).9901,(float)1.4938,(
          float)2.7177,(float)4.2454,(float)7.7199,(float)9.9831,(float)
```

```
10.2965,(float)10.0024,(float)7.7086,(float)4.2635,(float)1.2194,(

float).3056,(float)0.,(float)1.5,(float)0.,(float).2727,(float)
          .625,(float)1.0606,(float)1.6051,(float)2.7252,(float)4.2279,(

float)7.6637,(float)9.9322,(float)10.227[illegible],(float)9.9171,(float)

7.6573,(float)4.2184,(float)1.1961,(float).3133,(float)0.,(float)

2.,(float)0.,(float).2753,(float).6384,(float)1.1354,(float)1.664,

(float)2.7542,(float)3.9459,(float)7.1899,(float)9.2966,(float)

9.5868,(float)9.2911,(float)7.2048,(float)3.9474,(float)1.129,(
          float).2811,(float)0.,(float)4.,(float)0.,(float).2656,(float)
          .658,(float)1.189,(float)1.7299,(float)2.6718,(float)3.7776,(
          float)6.8236,(float)8.8635,(float)9.1188,(float)8.8535,(float)

6.8525,(float)3.7561,(float)1.0696,(float).2716,(float)0.,(float)
          6.,(float)0.,(float).2616,(float).6025,(float)1.0334,(float)

1.4472,(float)2.522,(float)3.7044,(float)6.7629,(float)8.7381,(
          float)9.0126,(float)8.7377,(float)6.7429,(float)3.7153,(float)

1.0442,(float).2684,(float)0.,(float)9.,(float)0.,(float).2567,(
          float).615,(float)1.0322,(float)1.4501,(float)2.5213,(float)

3.7166,(float)6.7689,(float)8.7283,(float)9.0069,(float)8.7303,(
          float)6.7754,(float)3.738,(float)1.0652,(float).2765,(float)0.
};
    static real cna2e[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)

160.,(float)170.,(float)180.,(float)0.,(float)0.,(float).1526,(

float).3299,(float).5186,(float).7355,(float)1.3117,(float)1.7864,

(float)2.6227,(float)3.3848,(float)3.5119,(float)3.3989,(float)

2.6088,(float)1.4323,(float).4123,(float).105,(float)0.,(float).5,
          (float)0.,(float).1466,(float).324,(float).5095,(float).7414,(
          float)1.3098,(float)1.8016,(float)2.6256,(float)3.3828,(float)
          3.5111,(float)3.3918,(float)2.6245,(float)1.437,(float).4156,(
          float).1115,(float)0.,(float).8,(float)0.,(float).1389,(float)
          .3278,(float).5399,(float).7499,(float)1.4611,(float)2.0041,(
          float)2.8421,(float)3.6682,(float)3.796,(float)3.6815,(float)

2.8486,(float)1.5607,(float).465,(float).1158,(float)0.,(float) 9,

(float)0.,(float).1547,(float).3536,(float).5956,(float).8675,(
          float)1.6577,(float)2.1614,(float)3.0455,(float)3.9357,(float)
          4.0622,(float)3.9195,(float)3.0225,(float)1.6694,(float).463,(

float).1169,(float)0.,(float)1.1,(float)0.,(float).1878,(float)
          .4249,(float).6982,(float)1.001,(float)1.6649,(float)2.3611,(
          float)3.619,(float)4.6821,(float)4.8318,(float)4.6926,(float)

3.6108,(float)1.4848,(float).56,(float).1396,(float)0.,(float)1.2,
```

```
(float)0.,(float).1929,(float).4189,(float).6926,(float)1.0156,(
          float)1.7242,(float)2.5269,(float)3.948,(float)5.0947,(float)
          5.254,(float)5.103,(float)3.9517,(float)2.1728,(float).6054,(

float).1699,(float)0.,(float)1.5,(float)0.,(float).2014,(float)

.412,(float).748,(float)1.0899,(float)1.7873,(float)2.5183,(float)

3.8838,(float)5.0173,(float)5.1797,(float)5.0229,(float)3.8697,(

float)2.1273,(float).6126,(float).173,(float)0.,(float)2.,(float)

0.,(float) 1915,(float).4388,(float).7244,(float)1.0354,(float)
          1.751,(float)2.4106,(float)3.6931,(float)4.774,(float)4.9262,(
          float)4.7731,(float)3.7039,(float)2.0259,(float).5808,(float)

.1458,(float)0.,(float)4.,(float)0.,(float).2038,(float).4157,(
          float).6921,(float)1.0066,(float)1.6656,(float)2.2891,(float)

3.4786,(float)4.4804,(float)4.6223,(float)4.481,(float)3.4656,(

float)1.8978,(float).5393,(float).1339,(float)0.,(float)6.,(float)
          0.,(float).1866,(float).3974,(float).6419,(float).9172,(float)

1.4828,(float)2.1442,(float)3.4153,(float)4.4316,(float)4.552,(
          float)4.4333,(float)3.4277,(float)1.8834,(float).5306,(float)

.1412,(float)0.,(float)20.,(float)0.,(float).1656,(float).3955,(
          float).6324,(float).8981,(float)1.4796,(float)2.1259,(float)

3.4177,(float)4.4451,(float)4.5763,(float)4.4204,(float)3.4285,(
          float)1.8721,(float).5357,(float).149,(float)0. };
    static integer icnm1e = 0;

    static real t, tstep;
    extern /* Subroutine */ int receive_real_32bit__(), tlu2ei_();
    static real estmch;
    extern /* Subroutine */ int send_real_32bit__();
    static real cne;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&estmch);
    if (tstep >= tapu) {
      tapu += dtapu;
      if (t < tstg2) {
          if (t < tstg1) {
            tlu2ei_(&estmch, &c_b2, cna1e, &icnm1e, &icna1e, &cne);
          } else {
            tlu2ei_(&estmch, &c_b2, cna2e, &icnm2e, &icna2e, &cne);
          }
      }
    }
    send_real_32bit__(&cne);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */
```

```
/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.17 Inerxyz.c

```
/* inerxyz.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__20 = 20;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real inerzz[20] = {
(float)3.141,(float)3.141,(float)3.141,(float)

3.141,(float)19.068,(float)20.526,(float)28.074,(float)34.176,(
          float)37.873,(float)40.583,(float)42.743,(float)46.409,(float)

46.883,(float)49.315,(float)136.726,(float)201.614,(float)261.952,
          (float)310.891,(float)346.848,(float)372.98 };
    static real ixx = (float)0.;
    static real iyy = (float)0.;
    static real izz = (float)0.;
    static integer itable = 0;
    static real tfinal = (float)62.501;
    static real masst1[20] = {
(float)2.403,(float)2.468,(float)2.524,(float)

2.724,(float)3.802,(float)4.231,(float)6.33,(float)8.631,(float)

11.024,(float)13.489,(float)14.932,(float)17.266,(float)18.955,(
          float)19.457,(float)22.121,(float)25.628,(float)30.806,(float)
          36.5,(float)40.889,(float)43.939 };
    static real inerxx[20] = {
(float).442,(float).442,(float).442,(float)
          .442,(float)1.03,(float)1.26,(float)2.007,(float)2.723,(float)

3.468,(float)4.109,(float)4.904,(float)5.49,(float)6.347,(float)

6.51,(float)8.169,(float)9.59,(float)11.776,(float)13.174,(float)
          15.196,(float)16.722 };
    static real ineryy[20] = {
(float)3.141,(float)3.141,(float)3.141,(float)

3.141,(float)19.068,(float)20.526,(float)28.074,(float)34.176,
          float)37.873,(float)40.583,(float)42.743,(float)46.409,(float)

46.883,(float)49.315,(float)136.726,(float)201.614,(float)261.952,
          (float)310.891,(float)346.848,(float)372.98 };

    static real mass, t;
    extern /* Subroutine */ int table_();
    static real tstep;
    extern /* Subroutine */ int receive_real_32bit__(),
send_real_32bit__();

/* initialize time */
```

```
    tstep = (float)0.;
    t = tstep * delt;
L10:
    send_real_32bit__(&iyy);
    receive_real_32bit__(&mass);
    table_(masst1, inerxx, &mass, &ixx, &c__20, &itable);
    table_(masst1, ineryy, &mass, &iyy, &c__20, &itable);
    table_(masst1, inerzz, &mass, &izz, &c__20, &itable);
    send_real_32bit__(&ixx);
    send_real_32bit__(&iyy);
    send_real_32bit__(&izz);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.18 Press.c

```
/* press.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__59 = 59;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real altt[59] = {
(float)0.,(float)2e3,(float)4e3,(float)6e3,(
          float)8e3,(float)1e4,(float)1.2e4,(float)1.4e4,(float)1.6e4,(

float)1.8e4,(float)2e4,(float)2.3e4,(float)2.6e4,(float)2.9e4,(

float)3.2e4,(float)3.5e4,(float)3.8e4,(float)4.1e4,(float)4.4e4,(

float)4.7e4,(float)5e4,(float)5.4e4,(float)5.8e4,(float)6.2e4,(

float)6.6e4,(float)7e4,(float)7.4e4,(float)7.8e4,(float)8.2e4,(
          float)8.6e4,(float)9e4,(float)9.5e4,(float)1e5,(float)1.05e5,(

float)1.1e5,(float)1.15e5,(float)1.2e5,(float)1.25e5,(float)1.3e5,
          (float)1.35e5,(float)1.4e5,(float)1.45e5,(float)1.5e5,(float)
          1.55e5,(float)1.6e5,(float)1.65e5,(float)1.7e5,(float)1.75e5,(

float)1.8e5,(float)1.9e5,(float)2e5,(float)2.1e5,(float)2.2e5,(

float)2.3e5,(float)2.4e5,(float)2.5e5,(float)2.75e5,(float)3e5,(
          float)999999. };
    static real presst[59] = {
(float)2116.25,(float)1967.72,(float)1827.78,(

float)1696.02,(float)1571.91,(float)1455.35,(float)1345.9,(float)

1243.2,(float)1147.72,(float)1057.49,(float)973.289,(float)857.26,
          (float)752.725,(float)658.783,(float)574.592,(float)499.356,(

float)432.644,(float)374.755,(float)324.623,(float)281.21,(float)
          243.614,(float)203.13,(float)166.178,(float)137.264,(float)
          113.389,(float)93.7284,(float)77.5639,(float)64.2586,(float)
          53.2944,(float)44.2494,(float)37.1196,(float)29.2323,(float)
          23.2726,(float)18.5578,(float)14.8375,(float)11.912,(float)
          9.60154,(float)7.76921,(float)6.30961,(float)5.14276,(float)
          4.20625,(float)3.45183,(float)2.84192,(float)2.34714,(float)
          1.94196,(float)1.60687,(float)1.32973,(float)1.10007,(float)
          .908378,(float).61551,(float).413455,(float).274355,(float)
          .178439,(float).113488,(float).070406,(float).042482,(float)
          .017169,(float).002643,(float).002643 };
    static integer itable = 0;

    static real t;
    extern /* Subroutine */ int table_();
```

```
    static real press, tstep;
    extern /* Subroutine */ int receive_real_32bit__(),
send_real_32bit__();
    static real alt;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    press = (float)0.;
L10:
    send_real_32bit__(&press);
    receive_real_32bit__(&alt);
    table_(altt, presst, &alt, &press, &c__59, &itable);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.19 Print.c

```
/* print.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real tprt = (float)0.;
    static real dtprt = (float)100.;

    /* Local variables */
    static real velocity, t, x, y, z, tstep;
    extern /* Subroutine */ int receive_real_32bit__();
    static real vrwmx, vrwmy, vrwmz;
    extern /* Subroutine */ int send_host_real__();
    static real garbage, alt, phi, psi, tht;

/*     INCLUDE ':pfp:include/target.for' */
    receive_real_32bit__(&garbage);
/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
/*
-----------------------------------------------------------------
-----c */
/* ---------------------------- main execution loop
--------------------c */
/*
-----------------------------------------------------------------
-----c */
L10:
/* --------------- receive parameters from partition #1
------------------c */
    receive_real_32bit__(&vrwmx);
    receive_real_32bit__(&vrwmy);
    receive_real_32bit__(&vrwmz);
    receive_real_32bit__(&phi);
    receive_real_32bit__(&tht);
    receive_real_32bit__(&psi);
    receive_real_32bit__(&x);
    receive_real_32bit__(&y);
    receive_real_32bit__(&z);
    receive_real_32bit__(&alt);
    if (tstep >= tprt) {
      velocity = sqrt(vrwmx * vrwmx + vrwmy * vrwmy + vrwmz * vrwmz);
      send_host_real__(&t);
      send_host_real__(&alt);
      send_host_real__(&x);
      send_host_real__(&y);
      send_host_real__(&z);
      send_host_real__(&velocity);
      send_host_real__(&vrwmx);
      send_host_real__(&vrwmy);
      send_host_real__(&vrwmz);
      send_host_real__(&phi);
```

```
      send_host_real__(&tht);
      send_host_real__(&psi);
      tprt += dtprt;
    }
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.20 Rho.c

```
/* rho.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__59 = 59;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real slglbm = (float)32.174048;
    static real tfinal = (float)62.501;
    static real altt[59] = {
(float)0.,(float)2e3,(float)4e3,(float)6e3,(
          float)8e3,(float)1e4,(float)1.2e4,(float)1.4e4,(float)1.6e4,(

float)1.8e4,(float)2e4,(float)2.3e4,(float)2.6e4,(float)2.9e4,(

float)3.2e4,(float)3.5e4,(float)3.8e4,(float)4.1e4,(float)4.4e4,(

float)4.7e4,(float)5e4,(float)5.4e4,(float)5.8e4,(float)6.2e4,(

float)6.6e4,(float)7e4,(float)7.4e4,(float)7.8e4,(float)8.2e4,(
          float)8.6e4,(float)9e4,(float)9.5e4,(float)1e5,(float)1.05e5,(

float)1.1e5,(float)1.15e5,(float)1.2e5,(float)1.25e5,(float)1.3e5,
          (float)1.35e5,(float)1.4e5,(float)1.45e5,(float)1.5e5,(float)
          1.55e5,(float)1.6e5,(float)1.65e5,(float)1.7e5,(float)1.75e5,(

float)1.8e5,(float)1.9e5,(float)2e5,(float)2.1e5,(float)2.2e5,(

float)2.3e5,(float)2.4e5,(float)2.5e5,(float)2.75e5,(float)3e5,(
          float)999999. };
    static real rhot[59] = {
(float)76474.,(float)72098.,(float)67917.,(float)

63925.,(float)60116.,(float)56483.,(float)53022.,(float)49725.,(
          float)46589.,(float)43606.,(float)40773.,(float)36790.,(float)

33113.,(float)29725.,(float)26610.,(float)23751.,(float)20794.,(
          float)18012.,(float)15602.,(float)13516.,(float)11709.,(float)

9670.1,(float)7987.,(float)6597.3,(float)5448.5,(float)4478.7,(
          float)3685.8,(float)3036.8,(float)2504.9,(float)2068.5,(float)

1710.,(float)1350.,(float)1067.6,(float)845.7,(float)664.7,(float)

524.12,(float)415.06,(float)330.06,(float)263.53,(float)211.22,(
          float)169.94,(float)137.22,(float)111.19,(float)90.4,(float)

74.713,(float)61.822,(float)51.159,(float)42.605,(float)35.578,(
          float)24.663,(float)16.957,(float)11.748,(float)8.0356,(float)
          5.3888,(float)3.5353,(float)2.263,(float).9171,(float).1488,(
          float).1488 };
    static integer itable = 0;
```

```
    static real rhod2, t;
    extern /* Subroutine */ int table_();
    static real tstep;
    extern /* Subroutine */ int receive_real_32bit__(),
send_real_32bit__();
    static real alt, rho;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    rho = (float)0.;
L10:
    receive_real_32bit__(&alt);
    table_(altt, rhot, &alt, &rho, &c__59, &itable);
    rho = rho * (float)1e-6 / slglbm;
    rhod2 = rho / (float)2.;
    send_real_32bit__(&rhod2);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.21 Shear.c

```
/* shear.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__59 = 59;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real altt[59] = {
(float)0.,(float)2e3,(float)4e3,(float)6e3,(
          float)8e3,(float)1e4,(float)1.2e4,(float)1.4e4,(float)1.6e4,(

float)1.8e4,(float)2e4,(float)2.3e4,(float)2.6e4,(float)2.9e4,(

float)3.2e4,(float)3.5e4,(float)3.8e4,(float)4.1e4,(float)4.4e4,(

float)4.7e4,(float)5e4,(float)5.4e4,(float)5.8e4,(float)6.2e4,(

float)6.6e4,(float)7e4,(float)7.4e4,(float)7.8e4,(float)8.2e4,(
          float)8.6e4,(float)9e4,(float)9.5e4,(float)1e5,(float)1.05e5,(

float)1.1e5,(float)1.15e5,(float)1.2e5,(float)1.25e5,(float)1.3e5,
          (float)1.35e5,(float)1.4e5,(float)1.45e5,(float)1.5e5,(float)
          1.55e5,(float)1.6e5,(float)1.65e5,(float)1.7e5,(float)1.75e5,(

float)1.8e5,(float)1.9e5,(float)2e5,(float)2.1e5,(float)2.2e5,(

float)2.3e5,(float)2.4e5,(float)2.5e5,(float)2.75e5,(float)3e5,(
          float)999999. };
    static real sheart[59] = {
(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)
          0.,(float)0.,(float)0. };
    static integer itable = 0;

    static real t;
    extern /* Subroutine */ int table_();
```

```
    static real shear, tstep;
    extern /* Subroutine */ int receive_real_32bit__(),
send_real_32bit__();
    static real alt;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
    shear = (float)0.;
L10:
    receive_real_32bit__(&alt);
    table_(altt, sheart, &alt, &shear, &c__59, &itable);
    send_real_32bit__(&shear);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.22 Target.c

```
/* target.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static doublereal delt = .001;
    static doublereal vtic[3] = { -6858.46,0.,-18411.68 };
    static doublereal tfinal = 62.501;
    static integer first1 = 1;
    static doublereal gmu = 1.4052477e16;
    static doublereal rtic[3] = { 22462673.6,0.,3781781.71 };

    /* System generated locals */
    doublereal d_1;

    /* Local variables */
    extern /* Subroutine */ int magt_();
    static doublereal mgrt;
    static integer i;
    static doublereal t, tdelt, mrtic, urtic[3], tstep;
    extern /* Subroutine */ int mvbys_(), receive_real_32bit__();
    static doublereal tl1, alt, grt[3];

/*
-------------------------------------------------------------------
------ */
/*      subroutine :          target(t,rtic,vtic) */
/*      function :            computes the rotational and translational
*/

/*                            target states */
/*      inputs :              t */
/*      outputs :             rtic,vtic */
/*
-------------------------------------------------------------------
------ */
/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&alt);
    magt_(rtic, &mrtic, urtic);
/* Computing 2nd power */
    d_1 = mrtic;
    mgrt = gmu / (d_1 * d_1);
    d_1 = -mgrt;
    mvbys_(&d_1, urtic, grt);
/*  integrate target acceleration and velocity */
    if (first1 == 1) {
      first1 = 0;
      tl1 = t;
    } else {
      tdelt = t - tl1;
      tl1 = t;
      for (i = 1; i <= 3; ++i) {
```

```
            rtic[i - 1] = rtic[i - 1] + vtic[i - 1] * tdelt + grt[i - 1] *
.5
                  * tdelt * tdelt;
            vtic[i - 1] += grt[i - 1] * tdelt;
/* L20: */
        }
    }
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.23 Vsnd.c

```
/* vsnd.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__59 = 59;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real altt[59] = {
(float)0.,(float)2e3,(float)4e3,(float)6e3,(
          float)8e3,(float)1e4,(float)1.2e4,(float)1.4e4,(float)1.6e4,(

float)1.8e4,(float)2e4,(float)2.3e4,(float)2.6e4,(float)2.9e4,(

float)3.2e4,(float)3.5e4,(float)3.8e4,(float)4.1e4,(float)4.4e4,(

float)4.7e4,(float)5e4,(float)5.4e4,(float)5.8e4,(float)6.2e4,(

float)6.6e4,(float)7e4,(float)7.4e4,(float)7.8e4,(float)8.2e4,(
          float)8.6e4,(float)9e4,(float)9.5e4,(float)1e5,(float)1.05e5,(

float)1.1e5,(float)1.15e5,(float)1.2e5,(float)1.25e5,(float)1.3e5,
          (float)1.35e5,(float)1.4e5,(float)1.45e5,(float)1.5e5,(float)
          1.55e5,(float)1.6e5,(float)1.65e5,(float)1.7e5,(float)1.75e5,(

float)1.8e5,(float)1.9e5,(float)2e5,(float)2.1e5,(float)2.2e5,(

float)2.3e5,(float)2.4e5,(float)2.5e5,(float)2.75e5,(float)3e5,(
          float)999999. };
    static real vsndt[59] = {
(float)1116.45,(float)1108.75,(float)1100.99,(

float)1093.19,(float)1085.32,(float)1077.4,(float)1069.43,(float)
          1061.39,(float)1053.3,(float)1045.15,(float)1036.93,(float)

1024.48,(float)1011.89,(float)999.14,(float)986.22,(float)973.14,(
          float)968.08,(float)968.08,(float)968.08,(float)968.08,(float)

968.08,(float)968.08,(float)968.08,(float)968.08,(float)968.19,(
          float)970.9,(float)973.59,(float)976.14,(float)978.95,(float)

987.62,(float)984.28,(float)987.59,(float)990.9,(float)994.18,(

float)1002.72,(float)1011.79,(float)1020.77,(float)1029.67,(float)
          1038.48,(float)1047.22,(float)1055.88,(float)1064.47,(float)
          1072.99,(float)1081.43,(float)1082.02,(float)1082.02,(float)
          1082.02,(float)1078.43,(float)1072.4,(float)1060.25,(float)

1047.98,(float)1025.62,(float)1000.11,(float)973.96,(float)947.12,
          (float)919.5,(float)884.,(float)894.5,(float)894.5 };
    static integer itable = 0;
```

```
    static real vsnd, t;
    extern /* Subroutine */ int table_();
    static real tstep;
    extern /* Subroutine */ int receive_real_32bit__(),
send_real_32bit__();
    static real alt;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&alt);
    table_(altt,  sndt, &alt, &vsnd, &c__59, &itable);
    send_real_32bit__(&vsnd);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.24 Vwind.c

```
/* vwind.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__59 = 59;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real tfinal = (float)62.501;
    static real altt[59] = {
(float)0.,(float)2e3,(float)4e3,(float)6e3,(
          float)8e3,(float)1e4,(float)1.2e4,(float)1.4e4,(float)1.6e4,(

float)1.8e4,(float)2e4,(float)2.3e4,(float)2.6e4,(float)2.9e4,(

float)3.2e4,(float)3.5e4,(float)3.8e4,(float)4.1e4,(float)4.4e4,(

float)4.7e4,(float)5e4,(float)5.4e4,(float)5.8e4,(float)6.2e4,(

float)6.6e4,(float)7e4,(float)7.4e4,(float)7.8e4,(float)8.2e4,(
          float)8.6e4,(float)9e4,(float)9.5e4,(float)1e5,(float)1.05e5,(

float)1.1e5,(float)1.15e5,(float)1.2e5,(float)1.25e5,(float)1.3e5,
          (float)1.35e5,(float)1.4e5,(float)1.45e5,(float)1.5e5,(float)
          1.55e5,(float)1.6e5,(float)1.65e5,(float)1.7e5,(float)1.75e5,(

float)1.8e5,(float)1.9e5,(float)2e5,(float)2.1e5,(float)2.2e5,(

float)2.3e5,(float)2.4e5,(float)2.5e5,(float)2.75e5,(float)3e5,(
          float)999999. };
    static real vwindt[59] = {
(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)
          0.,(float)0.,(float)0. };
    static integer itable = 0;

    static real t;
    extern /* Subroutine */ int table_();
```

```
    static real vwind, tstep;
    extern /* Subroutine */ int receive_real_32bit__(),
send_real_32bit__();
    static real alt;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&alt);
    table_(altt, vwindt, &alt, &vwind, &c__59, &itable);
    send_real_32bit__(&vwind);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.25 Windir.c

```
/* windir.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc   (in that order)
*/

#include "f2c.h"

/* Table of constant values */

static integer c__59 = 59;

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static real dtr = (float).017453292519943296;
    static real tfinal = (float)62.501;
    static real altt[59] = {
(float)0.,(float)2e3,(float)4e3,(float)6e3,(
          float)8e3,(float)1e4,(float)1.2e4,(float)1.4e4,(float)1.6e4,(

float)1.8e4,(float)2e4,(float)2.3e4,(float)2.6e4,(float)2.9e4,(

float)3.2e4,(float)3.5e4,(float)3.8e4,(float)4.1e4,(float)4.4e4,(

float)4.7e4,(float)5e4,(float)5.4e4,(float)5.8e4,(float)6.2e4,(

float)6.6e4,(float)7e4,(float)7.4e4,(float)7.8e4,(float)8.2e4,(
          float)8.6e4,(float)9e4,(float)9.5e4,(float)1e5,(float)1.05e5,(

float)1.1e5,(float)1.15e5,(float)1.2e5,(float)1.25e5,(float)1.3e5,
          (float)1.35e5,(float)1.4e5,(float)1.45e5,(float)1.5e5,(float)
          1.55e5,(float)1.6e5,(float)1.65e5,(float)1.7e5,(float)1.75e5,(

float)1.8e5,(float)1.9e5,(float)2e5,(float)2.1e5,(float)2.2e5,(

float)2.3e5,(float)2.4e5,(float)2.5e5,(float)2.75e5,(float)3e5,(
          float)999999. };
    static real windirt[59] = {
(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(

float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)

0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(float)0.,(
          float)0.,(float)0.,(float)0. };
    static integer itable = 0;

    /* Local variables */
```

```
    static real t;
    extern /* Subroutine */ int table_();
    static real cwdir, swdir, tstep;
    extern /* Subroutine */ int receive_real_32bit__();
    static real windir;
    extern /* Subroutine */ int send_real_32bit__();
    static real alt;

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&alt);
    table_(altt, windirt, &alt, &windir, &c__59, &itable);
    swdir = sin(windir * dtr);
    cwdir = cos(windir * dtr);
    send_real_32bit__(&swdir);
    send_real_32bit__(&cwdir);
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.3.26 Xcpe.c

```
/* xcpe.f -- translated by f2c (version of 3 February 1990  3:36:42).
   You must link the resulting object file with the libraries:
      -lF77 -lI77 -lm -lc    (in that order)
*/

#include "f2c.h"

/* Main program */ MAIN__()
{
    /* Initialized data */

    static real delt = (float).001;
    static integer icpa1e = 0;
    static integer icpm2e = 0;
    static integer icpa2e = 0;
    static real tapu = (float)0.;
    static real dtapu = (float)5.;
    static real tfinal = (float)62.501;
    static real tstg1 = (float)23.;
    static real tstg2 = (float)62.5;
    static real xcp11e[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(

float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(

float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)

160.,(float)170.,(float)180.,(float)0.,(float)40.2,(float)55.6,(

float)59.4,(float)58.9,(float)59.5,(float)59.3,(float)62.3,(float)
          67.1,(float)78.8,(float)85.2,(float)104.8,(float)112.6,(float)
          116.8,(float)134.4,(float)155.8,(float)158.4,(float).5,(float)

42.1,(float)55.,(float)61.,(float)57.9,(float)61.2,(float)61.4,(
          float)61.2,(float)67.8,(float)78.5,(float)86.1,(float)106.5,(

float)115.1,(float)118.3,(float)132.2,(float)156.5,(float)157.8,(

float).8,(float)41.1,(float)59.9,(float)59.4,(float)63.,(float)

62.,(float)63.4,(float)63.5,(float)71.8,(float)78.2,(float)88.,(

float)114.1,(float)125.4,(float)124.,(float)137.2,(float)159.5,(

float)161.1,(float).9,(float)40.3,(float)56.8,(float)62.2,(float)

63.8,(float)62.1,(float)64.4,(float)65.3,(float)72.2,(float)78.3,(

float)84.8,(float)117.8,(float)125.8,(float)127.7,(float)140.4,(
          float)166.9,(float)175.9,(float)1.1,(float)42.1,(float)58.6,(

float)60.3,(float)63.,(float)63.5,(float)63.8,(float)64.5,(float)
          69.,(float)79.9,(float)85.7,(float)123.4,(float)130.3,(float)
          130.9,(float)147.6,(float)188.,(float)197.5,(float)1.2,(float)

43.5,(float)55.8,(float)60.3,(float)63.6,(float)63.2,(float)63.3,(

float)68.3,(float)74.2,(float)80.6,(float)86.,(float)127.3,(float)

131.4,(float)134.,(float)152.3,(float)189.7,(float)198.3,(float)
```

```
1.5,(float)43.5,(float)57.6,(float)59.5,(float)62.3,(float)61.4,(
float)66.9,(float)72.5,(float)76.9,(float)82.8,(float)85.7,(float)
95.2,(float)101.9,(float)105.4,(float)111.5,(float)147.6,(float)
160.2,(float)2.,(float)46.,(float)59.3,(float)63.7,(float)65.4,(
float)67.,(float)73.3,(float)75.4,(float)81.2,(float)82.6,(float)
86.,(float)101.6,(float)104.8,(float)107.8,(float)109.7,(float)
129.6,(float)145.7,(float)4.,(float)51.3,(float)60.2,(float)68.3,(
float)69.9,(float)72.1,(float)77.9,(float)79.9,(float)80.9,(float)
84.2,(float)86.6,(float)113.9,(float)116.3,(float)119.3,(float)
121.2,(float)125.9,(float)134.9,(float)6.,(float)52.3,(float)64.2,
(float)66.4,(float)69.,(float)68.,(float)74.3,(float)75.5,(float)
          81.9,(float)82.5,(float)86.1,(float)117.2,(float)120.7,(float)
124.,(float)126.6,(float)128.8,(float)134.4,(float)9.,(float)50.8,
          (float)62.6,(float)66.6,(float)70.1,(float)68.5,(float)73.2,(
          float)76.6,(float)81.3,(float)83.3,(float)86.4,(float)117.3,(
          float)122.6,(float)125.2,(float)126.,(float)129.8,(float)135.2
};
    static real xcp12e[205] = {
(float)11.,(float)16.,(float)0.,(float)5.,(
float)10.,(float)15.,(float)20.,(float)30.,(float)40.,(float)60.,(
float)80.,(float)90.,(float)100.,(float)120.,(float)140.,(float)
160.,(float)170.,(float)180.,(float)0.,(float)33.5,(float)35.,(
float)35.3,(float)36.4,(float)36.8,(float)38.,(float)37.9,(float)
42.4,(float)45.2,(float)45.6,(float)52.6,(float)59.,(float)59.6,(
float)69.4,(float)79.8,(float)82.9,(float).5,(float)35.2,(float)
33.4,(float)33.3,(float)36.4,(float)35.5,(float)38.4,(float)40.4,(
float)42.9,(float)45.4,(float)47.7,(float)51.4,(float)57.2,(float)
59.3,(float)68.1,(float)80.7,(float)80.1,(float).8,(float)34.4,(
float)34.8,(float)35.1,(float)36.6,(float)38.5,(float)40.3,(float)
41.4,(float)43.5,(float)46.6,(float)47.6,(float)53.5,(float)63.1,(
float)63.5,(float)71.1,(float)80.8,(float)83.7,(float).9,(float)
35.7,(float)36.,(float)35.1,(float)35.8,(float)38.2,(float)40.9,(
float)42.8,(float)44.3,(float)46.7,(float)45.8,(float)53.6,(float)
65.1,(float)67.,(float)71.7,(float)86.8,(float)90.6,(float)1.1,(
float)37.9,(float)38.6,(float)36.4,(float)37.4,(float)38.4,(float)
```

```
41.2,(float)43.6,(float)42.8,(float)43.6,(float)47.9,(float)56.7,(
float)67.4,(float)67.6,(float)77.,(float)96.2,(float)102.6,(float)
1.2,(float)36.7,(float)35.1,(float)36.5,(float)35.,(float)40.,(
float)41.7,(float)41.1,(float)43.,(float)46.7,(float)47.7,(float)
59.2,(float)66.5,(float)67.6,(float)80.,(float)97.9,(float)102.7,(
float)1.5,(float)34.7,(float)35.2,(float)36.7,(float)39.5,(float)
40.5,(float)40.9,(float)41.7,(float)45.,(float)45.7,(float)47.5,(
float)49.5,(float)53.5,(float)55.5,(float)59.3,(float)75.8,(float)
          82.4,(float)2.,(float)35.,(float)36.1,(float)37.,(float)38.4,(
float)40.5,(float)41.,(float)41.,(float)44.4,(float)45.5,(float)
47.6,(float)52.6,(float)54.3,(float)55.6,(float)58.,(float)67.3,(
float)76.5,(float)4.,(float)33.7,(float)36.4,(float)37.3,(float)
39.7,(float)41.,(float)43.6,(float)41.5,(float)44.4,(float)45.7,(
float)46.4,(float)53.8,(float)60.6,(float)60.8,(float)62.8,(float)
65.2,(float)70.,(float)6.,(float)37.8,(float)36.2,(float)37.6,(
float)39.9,(float)39.6,(float)42.2,(float)43.4,(float)43.,(float)
46.9,(float)48.7,(float)54.3,(float)62.4,(float)65.1,(float)64.3,(
float)68.2,(float)70.5,(float)20.,(float)35.,(float)36.1,(float)
37.9,(float)39.5,(float)40.3,(float)40.7,(float)42.,(float)42.5,(
float)46.8,(float)48.7,(float)52.8,(float)62.,(float)64.5,(float)
          65.1,(float)67.2,(float)68.8 };
    static integer icpm1e = 0;

    static real xcpe, t, tstep;
    extern /* Subroutine */ int receive_real_32bit__(), tlu2ei_();
    static real alfate, estmch;
    extern /* Subroutine */ int send_real_32bit__();

/* initialize time */
    tstep = (float)0.;
    t = tstep * delt;
L10:
    receive_real_32bit__(&estmch);
    receive_real_32bit__(&alfate);
    if (tstep >= tapu) {
      tapu += dtapu;
      if (t < tstg2) {
          if (t < tstg1) {
            tlu2ei_(&estmch, &alfate, xcpl1e, &icpm1e, &icpa1e, &xcpe);
          } else {
            tlu2ei_(&estmch, &alfate, xcpl2e, &icpm2e, &icpa2e, &xcpe);
          }
      }
    }
    send_real_32bit__(&xcpe);
```

```
/* increment time */
    tstep += (float)1.;
    t = tstep * delt;
    if (t < tfinal) {
      goto L10;
    }
} /* MAIN__ */

/* Main program alias */ int main_ () { MAIN__ (); }
```

A.4 Crossbar Code

```
boost2a                 is    boost2a.fpx       on x13
boost2a1          is    boost2a1.fpx      on x14
boost2a3          is    boost2a3.fpx      on x15
boost2c2          is    boost2c2.fpx      on y11

boost2b                 is    boost2b.fpp       on x0
boost2c                 is    boost2c.fpp       on x1
attlm             is    attlm.fpp         on x2
cg123             is    cg123.fpp         on x3
inerxyz                 is    inerxyz.fpp       on x4
cne                     is    cne.fpp                 on x5
xcpe              is    xcpe.fpp          on x6
press             is    press.fpp         on x7
vsnd              is    vsnd.fpp          on x8
rho                     is    rho.fpp                 on x9
vwind             is    vwind.fpp         on x10
windir                  is    windir.fpp        on x11
shear             is    shear.fpp         on x12
boost2a2          is    boost2a2.fpp      on y0
aeroca                  is    aeroca.fpp        on y1
aerocn                  is    aerocn.fpp        on y2
aeroxcp                 is    aeroxcp.fpp       on y5
boost2c1          is    boost2c1.fpp      on y9
bauto             is    bauto.fpp         on y10
boost2b1          is    boost2b1.fpp      on y12
print             is    print.fpp         on y13
timer             is    timer.fpp         on y15


cycle timer.2[garbage]          -->   print;

loop

[0:]
cycle cg123.2[cg(1)]                  -->   boost2b boost2c1 bauto
boost2b1 timer;
cycle cg123.2[cg(2)]                  -->   boost2b boost2c1 bauto
boost2b1 timer;
cycle cg123.2[cg(3)]                  -->   boost2b boost2c1 bauto
boost2b1 timer;
cycle press.2[press]                  -->   boost2b1 timer;
[cycle] boost2a1.2[mach]              -->   boost2b [timer];
cycle boost2a1.2[qa]                  -->   boost2b timer;
[cycle] boost2a3.2[p]                 -->   boost2c1 [timer];
[10:]
cycle boost2a3.2[q]                   -->   boost2c1 timer;
cycle boost2a3.2[r]                   -->   boost2c1 timer;
cycle boost2a.2[ud]                   -->   boost2c1 timer;
cycle boost2a.2[vd]                   -->   boost2c1 timer;
cycle boost2a.2[wd]                   -->   boost2c1 timer;
[20:]
cycle boost2a3.2[pd]                  -->   boost2c1 boost2a2 timer;
cycle boost2a3.2[qd]                  -->   boost2c1 boost2a2 timer;
cycle boost2a3.2[rd]                  -->   boost2c1 boost2a2 timer;
[cycle] boost2a.2[gr(1)]              -->   boost2c [timer];
cycle boost2a.2[gr(2)]          -->   boost2c timer;
cycle boost2a.2[gr(3)]          -->   boost2c timer;
[30:]
cycle boost2c2.2[mvs]                 -->   boost2c timer;
cycle boost2c2.2[uvs(1)]              -->   boost2c timer;
cycle boost2c2.2[uvs(2)]              -->   boost2c timer;
```

```
cycle boost2c2.2[uvs(3)]                --> boost2c timer;
cycle boost2c.2[at(1)]          --> boost2c2 timer;
[40:]
cycle boost2c.2[at(2)]          --> boost2c2 timer;
cycle boost2c.2[at(3)]          --> boost2c2 timer;
cycle boost2c.2[delxd]          --> boost2c2 timer;
cycle boost2c.2[delyd]          --> boost2c2 timer;
cycle boost2c.2[delzd]          --> boost2c2 timer;
[cycle] inerxyz.2[iyy]                  --> bauto [timer];
[50:]
cycle boost2a.2[alt]                    --> bauto timer;
cycle boost2a1.2[vrwm(1)]               --> bauto print timer;
cycle boost2a1.2[vrwm(2)]               --> bauto print timer;
cycle boost2a1.2[vrwm(3)]               --> bauto print timer;
[cycle] boost2a.2[mass]                 --> cg123 inerxyz boost2a3
[timer];
cycle boost2a.2[alt]                    --> press vsnd rho vwind windir
shear timer;
[60:]
cycle boost2a.4[xyz(1)]         --> boost2a1 boost2a3 timer;
cycle boost2a.4[xyz(2)]         --> boost2a1 boost2a3 timer;
cycle boost2a.4[xyz(3)]         --> boost2a1 boost2a3 timer;
cycle boost2a.4[xyzd(1)]                --> boost2a1 timer;
cycle boost2a.4[xyzd(2)]                --> boost2a1 timer;
[80:]
cycle boost2a.4[xyzd(3)]                --> boost2a1 timer;
cycle boost2a2.2[cim(1)]                --> boost2a1 boost2a boost2a3
timer;
cycle boost2a2.2[cim(2)]                --> boost2a1 boost2a boost2a3
timer;
cycle boost2a2.2[cim(3)]                --> boost2a1 boost2a boost2a3
timer;
[90:]
cycle boost2a2.2[cim(4)]                --> boost2a1 boost2a boost2a3
timer;
cycle boost2a2.2[cim(5)]                --> boost2a1 boost2a boost2a3
timer;
cycle boost2a2.2[cim(6)]                --> boost2a1 boost2a boost2a3
timer;
cycle boost2a2.2[cim(7)]                --> boost2a1 boost2a boost2a3
timer;
cycle boost2a2.2[cim(8)]                --> boost2a1 boost2a boost2a3
timer;
[100:]
cycle boost2a2.2[cim(9)]                --> boost2a1 boost2a boost2a3
timer;

cycle boost2a2.2[phi]                   --> print timer;
cycle boost2a2.2[tht]                   --> print timer;
cycle boost2a2.2[psi]                   --> print timer;

cycle boost2b1.2[fxt]                   --> boost2a boost2a3 timer;
[cycle] vwind.2[vwind]                  --> boost2a1 [timer];
      [110:]
cycle boost2b1.2[fyt]                   --> boost2a boost2a3 timer;
[cycle] shear.2[shear]                  --> boost2a1 [timer];
cycle boost2b1.2[fzt]                   --> boost2a boost2a3 timer;
[cycle] windir.2[swdir]                 --> boost2a1 [timer];
cycle boost2b1.2[mxt]                   --> boost2a3 timer;
[cycle] windir.2[cwdir]                 --> boost2a1 [timer];
[110:]
cycle boost2b1.2[myt]                   --> boost2a3 timer;
cycle boost2b1.2[mzt]                   --> boost2a3 timer;
      [120:]
```

```
cycle boost2b.2[frcx]                --> boost2a boost2a3 timer;
cycle boost2b.2[frcy]                --> boost2a boost2a3 timer;
cycle boost2b.2[frcz]                --> boost2a boost2a3 timer;
[120:]
cycle boost2b.2[mrcx]                --> boost2a3 timer;
cycle boost2b.2[mrcy]                --> boost2a3 timer;
      [130:]
cycle boost2b.2[mrcz]                --> boost2a3 timer;
[cycle] boost2b1.2[mdott]            --> boost2a [timer];
cycle boost2b.2[mdotf]         --> boost2a timer;
[cycle] boost2c1.2[delphi]           --> boost2c [timer];
cycle boost2c1.2[deltht]             --> boost2c timer;
[130:]
cycle boost2c1.2[delpsi]             --> boost2c timer;
cycle boost2c1.2[delu]         --> boost2c timer;
      [140:]
cycle boost2c1.2[delv]         --> boost2c timer;
cycle boost2c1.2[delw]         --> boost2c timer;
[cycle] bauto.2[cmmd(1)]             --> boost2b1 [timer];
cycle bauto.2[cmmd(2)]         --> boost2b1 timer;
[140:]
cycle bauto.2[dlpc]                  --> boost2b timer;
cycle bauto.2[dlyc]                  --> boost2b timer;
      [150:]
cycle bauto.2[sq]                    --> boost2b timer;
cycle bauto.2[sr]                    --> boost2b timer;
cycle bauto.2[mdltfr]                --> boost2b timer;
[150:]
cycle bauto.2[malpha]                --> boost2b timer;
cycle bauto.2[estmch]                --> cne xcpe timer;
      [160:]
cycle bauto.2[alfate]                --> xcpe timer;
cycle cne.2[cne]                     --> bauto timer;
cycle xcpe.2[xcpe]                   --> bauto timer;
[cycle] boost2c.2[pm(1)]             --> boost2b [timer];
[160]
cycle boost2c.2[pm(2)]         --> boost2b timer;
cycle boost2c.2[pm(3)]         --> boost2b timer;
[cycle] cg123.2[cg(1)]               --> boost2a3 [timer];
      [170:]
cycle cg123.2[cg(2)]                 --> boost2a3 timer;
cycle cg123.2[cg(3)]                 --> boost2a3 timer;
cycle inerxyz.2[ixx]                 --> boost2a3 timer;
[170:]
cycle inerxyz.2[iyy]                 --> boost2a3 timer;
cycle inerxyz.2[izz]                 --> boost2a3 timer;
      [180:]
cycle vsnd.2[vsnd]                         --> boost2a1 timer;
cycle boost2a1.2[mach]         --> aeroca aerocn aeroxcp timer;
cycle boost2a1.2[alfat]        --> aeroca aerocn aeroxcp timer;
[180:]
cycle rho.2[rhod2]                   --> boost2a1 timer;
cycle aeroca.2[ca]                   --> boost2a1 timer;
      [190:]
cycle aerocn.2[cn]                   --> boost2a1 timer;
cycle boost2a1.2[fxa]                --> boost2a boost2a3 timer;
cycle boost2a1.2[fya]                --> boost2a boost2a3 timer;
[190:]
cycle boost2a1.2[fza]                --> boost2a boost2a3 timer;
cycle aeroxcp.2[xcp]                 --> boost2a3 timer;
[cycle] attlm.2[attlm]               --> boost2c [timer];
      [200:]
cycle boost2c.2[psier]         --> bauto timer;
cycle boost2c.2[thter]         --> bauto timer;
```

```
cycle boost2c.2[sq]                --> bauto timer;
[200]
cycle boost2c.2[sr]                --> bauto timer;
[cycle] boost2a.2[x]               --> print [timer];
cycle boost2a.2[y]                      --> print timer;
      [210:]
cycle boost2a.2[z]                      --> print timer;
cycle boost2a.2[alt]               --> print timer;
```